

\$22.00

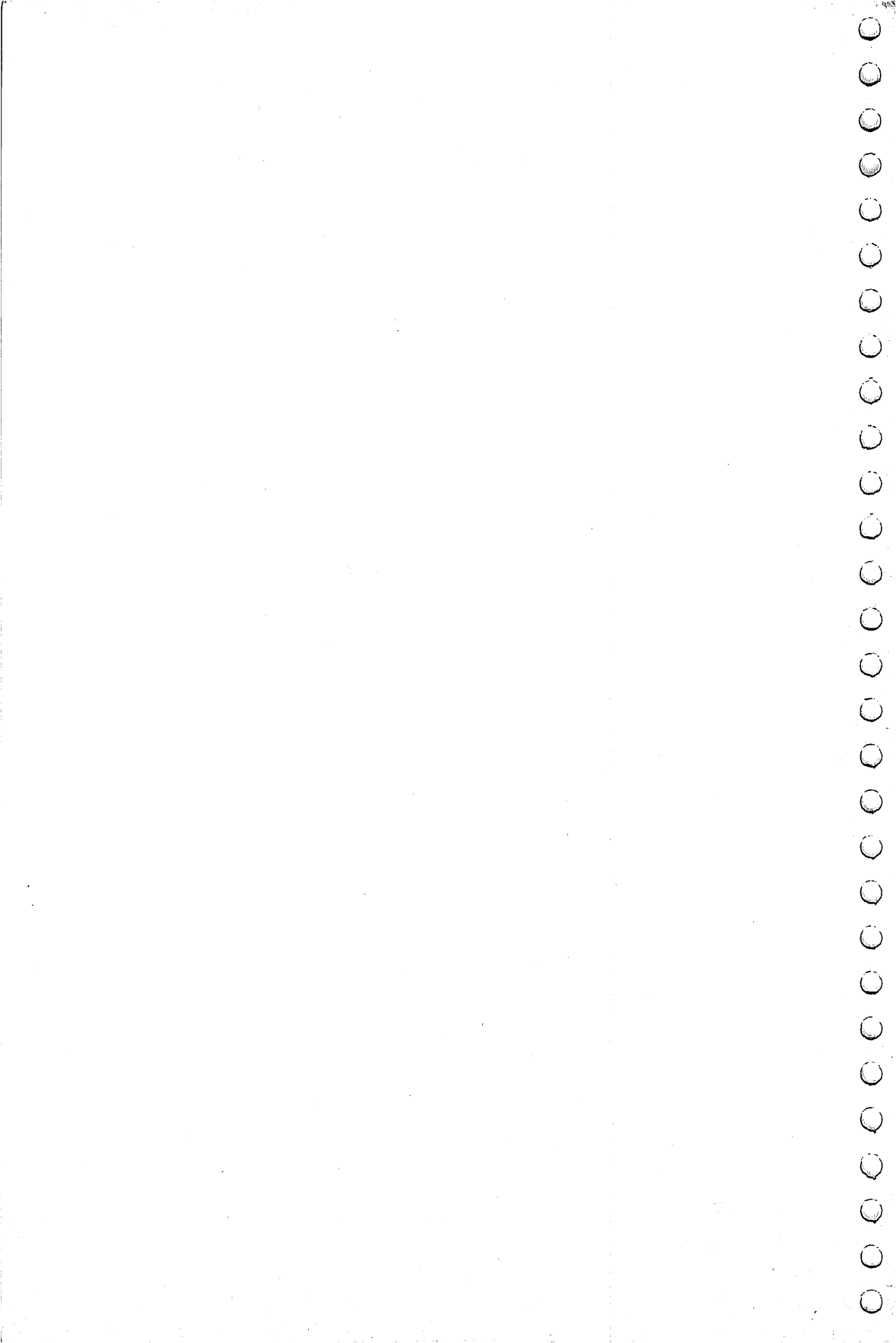
# Hands-On BASIC

FOR THE  
COMMODORE  
64

A BYTE BOOK

**HERBERT PECKHAM**

with WADE ELLIS, JR.  
and ED LODI





# **HANDS-ON BASIC**

FOR THE COMMODORE 64

## OTHER COMPUTER LITERACY TITLES

- |                                      |   |
|--------------------------------------|---|
| <b>Luehrmann &amp; Peckham</b>       | <b>Apple Pascal: A Hands-On Approach</b><br>(McGraw-Hill, 1981)                         |
| <b>Peckham</b>                       | <b>BASIC: A Hands-On Method</b><br>(McGraw-Hill, 1978)                                  |
| <b>Luehrmann &amp; Peckham</b>       | <b>Computer Literacy: A Hands-On Approach</b><br>(Apple II version) (McGraw-Hill, 1983) |
| <b>Luehrmann &amp; Peckham</b>       | <b>Computer Literacy: A Hands-On Approach</b><br>(TRS-80 version) (McGraw-Hill, 1983)   |
| <b>Luehrmann &amp; Peckham</b>       | <b>A Computer Literacy Survival Kit</b><br>(Apple II version) (McGraw-Hill, 1984)       |
| <b>Peckham with Ellis &amp; Lodi</b> | <b>Hands-On BASIC for the Apple II</b><br>(McGraw-Hill, 1982)                           |
| <b>Peckham with Ellis &amp; Lodi</b> | <b>Hands-On BASIC for Atari Computers</b><br>(McGraw-Hill, 1983)                        |
| <b>Peckham</b>                       | <b>Hands-On BASIC for the IBM Personal Computer</b><br>(McGraw-Hill, 1982)              |
| <b>Peckham with Ellis &amp; Lodi</b> | <b>Hands-On BASIC for the TI 99/4A</b><br>(McGraw-Hill, 1983)                           |
| <b>Peckham with Ellis &amp; Lodi</b> | <b>Hands-On BASIC for the TRS-80 Color Computer</b><br>(McGraw-Hill, 1983)              |
| <b>Peckham</b>                       | <b>Hands-On BASIC with a PET</b><br>(McGraw-Hill, 1979)                                 |
| <b>Peckham</b>                       | <b>Programming BASIC with the TI Home Computer</b><br>(McGraw-Hill, 1980)               |



# **HANDS-ON BASIC**

## **FOR THE COMMODORE 64**

**HERBERT PECKHAM**

with **WADE ELLIS, Jr.**

and **ED LODI**

**Computer  
Literacy**

A BOOK

McGRAW-HILL BOOK COMPANY

NEW YORK ST. LOUIS SAN FRANCISCO AUCKLAND  
BOGOTÁ HAMBURG JOHANNESBURG LONDON MADRID  
MEXICO MONTREAL NEW DELHI PANAMA PARIS  
SÃO PAULO SINGAPORE SIDNEY TOKYO TORONTO



## **HANDS-ON BASIC FOR THE COMMODORE 64**

Copyright © 1984 by McGraw-Hill, Inc. All rights reserved. Printed in the United States of America. Except as permitted under the United States Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without the prior written permission of the publisher.

234567890 HALHAL 8987654

ISBN 0-07-049154-2

The editor was Debbie K. Dennis;  
the production supervisor was Joe Campanella.  
Halliday Lithograph Corporation was printer and binder.

### **Library of Congress Cataloging in Publication Data**

Peckham, Herbert D.

Hands-on BASIC for the Commodore 64.

(A Computer literacy book)

Includes index.

1. Commodore 64 (Computer)—Programming.
  2. Basic (Computer program language) I. Ellis, Wade. II. Lodi, Ed. III. Title. IV. Title: Hands-on B.A.S.I.C. for the Commodore 64. V. Series.
- QA76.8.C64P43 1984 001.64'2 83-26808  
ISBN 0-07-049154-2

# CONTENTS

<b>PREFACE</b>	1
Acknowledgments	2

<b>INTRODUCTION</b>	5
What Is BASIC?	5
Where Did BASIC Originate?	6
How to Use This Book	6

## **CHAPTER 1**

### **GETTING ACQUAINTED WITH YOUR COMMODORE 64 COMPUTER**

1-1	<b>OBJECTIVES</b>	9
	Learning to Start the Computer	9
	Using Direct Mode	9
	Correcting Mistakes	10
1-2	<b>DISCOVERY EXERCISES</b>	10
1-3	<b>DISCUSSION</b>	15
	Learning to Start the Computer	15
	Using Direct Mode	16
	Correcting Mistakes	17
1-4	<b>PRACTICE TEST</b>	18

## CHAPTER 2

### INTRODUCTION TO BASIC 21

2—1	OBJECTIVES	21
	Learning to Edit Programs	21
	Learning the Requirements for BASIC Programs	21
	Telling the Computer What to Do	21
	Entering and Controlling Programs	21
	Assigning Variable Names in BASIC	22
2—2	DISCOVERY EXERCISES	22
2—3	DISCUSSION	31
	Learning to Edit Programs	31
	Learning the Requirements For BASIC Programs	31
	Telling the Computer What to Do	33
	Entering and Controlling Programs	34
	Assigning Variable Names in BASIC	34
2—4	PRACTICE TEST	37

## CHAPTER 3

### COMPUTER ARITHMETIC AND PROGRAM MANAGEMENT 41

3—1	OBJECTIVES	41
	Doing Arithmetic on the Computer	41
	Using Parentheses in Computations	41
	Learning E Notation for Numbers	41
	Initializing a Diskette	42
	Storing and Retrieving Programs	42
3—2	DISCOVERY EXERCISES	42
3—3	DISCUSSION	51
	Doing Arithmetic on the Computer	51
	Using Parentheses in Computations	53
	Learning E Notation for Numbers	54
	Initializing a Diskette	55
	Storing and Retrieving Programs	55
3—4	PRACTICE TEST	56

**CHAPTER 4****INPUT, OUTPUT, AND SIMPLE APPLICATIONS 61**

4—1	OBJECTIVES	61
	Getting Numbers into a BASIC Program	61
	Printing Out Variables and Strings	61
	Spacing the Printout	61
	Using the REM Statement	61
	Working with Program Examples	62
4—2	DISCOVERY EXERCISES	62
4—3	DISCUSSION	72
	Getting Numbers into a BASIC Program	72
	Printing out Variables and Strings	74
	Spacing the Printout	75
	Using the REM Statement	77
4—4	WORKING WITH PROGRAM EXAMPLES	78
	Example 1 - Unit Prices	78
	Example 2 - Converting Temperature	80
	Example 3 - Sum and Product of Numbers	81
4—5	PROBLEMS	83
4—6	PRACTICE TEST	89

**CHAPTER 5****DECISIONS, BRANCHING, AND APPLICATIONS 93**

5—1	OBJECTIVES	93
	Understanding Transfer Statements	93
	Working with Program Examples	93
	Finding Errors in Programs	93
5—2	DISCOVERY EXERCISES	94
5—3	DISCUSSION	101
	Understanding Transfer Statements	101
5—4	WORKING WITH PROGRAM EXAMPLES	104
	Example 1 - Printout of Number Patterns	104
	Example 2 - Automobile License Fees	106
	Example 3 - Averaging Numbers	111
5—5	FINDING ERRORS IN PROGRAMS	113
5—6	PROBLEMS	113
5—7	PRACTICE TEST	118



**CHAPTER 6****LOOPING AND FUNCTIONS 121**

- 6—1 OBJECTIVES 121
  - Understanding Built-in Looping 121
  - Using Built-in Functions 121
  - Working with Program Examples 121
- 6—2 DISCOVERY EXERCISES 121
- 6—3 DISCUSSION 130
  - Understanding Built-in Looping 130
  - Using Built-in Functions 134
- 6—4 WORKING WITH PROGRAM EXAMPLES 137
  - Example 1 - Finding the Average of a Group of Numbers 137
  - Example 2 - Temperature Conversion Table 139
  - Example 3 - Exact Division 140
  - Example 4 - Depreciation Schedule 141
- 6—5 PROBLEMS 144
- 6—6 PRACTICE TEST 148

**CHAPTER 7****WORKING WITH COLLECTIONS OF NUMBERS 151**

- 7—1 OBJECTIVES 151
  - Using Single- and Double-Subscripted Variables 151
  - Saving Space for Arrays 151
  - Using Subscripted Variables and FOR NEXT loops 151
  - Working with Program Examples 151
- 7—2 DISCOVERY EXERCISES 152
- 7—3 DISCUSSION 161
  - Using Single- and Double-Subscripted Variables 161
  - Saving Space for Arrays 163
  - Using Subscripted Variables and FOR NEXT Loops. 164
- 7—4 WORKING WITH PROGRAM EXAMPLES 165
  - Example 1 - Examination Grades 165
  - Example 2 - Course Grades 168
  - Example 3 - Array Operations 171
- 7—5 PROBLEMS 173
- 7—6 PRACTICE TEST 178

## CHAPTER 8

### STRING VARIABLES 181

- 8—1 OBJECTIVES 181
  - Understanding String Input and Output 181
  - Using String Functions 181
  - Working with Program Examples 181
- 8—2 DISCOVERY EXERCISES 181
- 8—3 DISCUSSION 191
  - Understanding String Input and Output 191
  - Using String Functions 192
- 8—4 WORKING WITH PROGRAM EXAMPLES 193
  - Example 1 - String Reversal 193
  - Example 2 - Word Count 194
  - Example 3 - Replacement Code 194
  - Example 4 - A Sales Chart as a String Array 195
- 8—5 PROBLEMS 198
- 8—6 PRACTICE TEST 199

## CHAPTER 9

### "DO-IT-YOURSELF" FUNCTIONS AND SUBROUTINES 201

- 9—1 OBJECTIVES 201
  - Defining "Do-It-Yourself" Functions 201
  - Understanding Subroutines 201
  - Working with Program Examples 201
- 9—2 DISCOVERY EXERCISES 201
- 9—3 DISCUSSION 209
  - Defining "Do-It-Yourself" Functions 209
  - Understanding Subroutines 210
- 9—4 WORKING WITH PROGRAM EXAMPLES 212
  - Example 1 - Rounding Off Dollar Values to Cents 212
  - Example 2 - Carpet Estimating 214
- 9—5 PROBLEMS 219
- 9—6 PRACTICE TEST 221

<b>CHAPTER 10</b>	
<b>CHARACTER GRAPHICS AND COLOR</b>	<b>225</b>
10—1 OBJECTIVES	225
Controlling Programs	225
Placing Characters	225
Working with the Graphics Features of the Commodore 64	225
Controlling Color	225
Working with Program Examples	225
10—2 DISCOVERY EXERCISES	226
10—3 DISCUSSION	235
Controlling Programs	235
Placing Characters	236
Working with the Graphics Features of the Commodore 64	237
Controlling Color	238
10—4 WORKING WITH PROGRAM EXAMPLES	239
Example 1 - Cursor Movement	239
Example 2 - Bar Graphs	241
Example 3 - Color Bar Graphs	241
Example 4 - Animation	242
10—5 PROBLEMS	243
10—6 PRACTICE TEST	244
<b>CHAPTER 11</b>	
<b>RANDOM NUMBERS AND SIMULATIONS</b>	<b>247</b>
11—1 OBJECTIVES	247
Understanding Random-Number Generators	247
Designing Sets of Random Numbers	247
Working with Program Examples	247
11—2 DISCOVERY EXERCISES	248
Setting Up the Random-Number Generator	248
11—3 DISCUSSION	252
Understanding Random-Number Generators	252
Designing Sets of Random Numbers	253

11—4	WORKING WITH PROGRAM EXAMPLES	254
	Example 1 - Flipping Coins	254
	Example 2 - Random Integers	256
	Example 3 - Distribution of Random Numbers	256
	Example 4 - Random Walk	257
	Example 5 - Random Colors	259
	Example 6 - Birthday Pairs in a Crowd	259
11—5	PROBLEMS	261
11—6	PRACTICE TEST	262

**CHAPTER 12****FILES** 265

12—1	OBJECTIVES	265
	Opening and Closing Information Paths	265
	Writing Information to a File	265
	Retrieving Information from a File	265
	Working with Program Examples	265
12—2	DISCOVERY EXERCISES	265
12—3	DISCUSSION	272
	Opening and Closing Information Paths	272
	Writing Information to a File	273
	Retrieving Information from a File	274
12—4	PROGRAM EXAMPLES	275
	Example 1 - Mail List Data Entry Program	276
	Example 2 - Add Records Program	277
	Example 3 - Mailing Label Program	278
	Example 4 - Selected Labels Program	279
	Example 5 - Modifying the MAILLIST File	280
	Example 6 - Menu-Driven Mailing Program	282
	Example 7 - Precision Record Access	284
12—5	PROBLEMS	285
12—6	PRACTICE TEST	286

**APPENDIX A**

**OTHER CONFIGURATIONS 289**

Commodore 64 without a Disk Drive 289

Commodore 64 with a Commodore Datacassette  
Recorder 289

**APPENDIX B**

**SUPPORT PROGRAM 291**

The C-64 Wedge or DOS Support Program 291

**APPENDIX C**

**GLOSSARY 293**

**APPENDIX D**

**PRACTICE TEST SOLUTIONS 297**

**APPENDIX E**

**SOLUTIONS TO ODD-NUMBERED PROBLEMS 307**

**INDEX 325**



# PREFACE

This book is a modification of *BASIC: A Hands-on Method*, which introduces students to BASIC on a number of different time-sharing computers. The earlier book has been revised and modified to be used specifically on a personal computer manufactured by Commodore Business Machines, Inc. The thinking behind and justification for the original work remain unchanged and bear repeating.

Most BASIC programming texts have two serious drawbacks. First, almost all the texts presuppose a knowledge of mathematics that most of our intended readers do not have. Second, most texts require readers to spend little, if any, time on the computer. Typically, students try to study programming like any other subject and do not experiment with or execute programs on the computer. Our experience indicates that people understand text material better and more rapidly when it is preceded by a good deal of hands-on experimentation.

Most textbooks are used in classrooms, and certainly many people learn programming in this traditional setting. However, personal computers will soon be in such widespread use that many people will learn programming outside the classroom. This text has been designed for anyone, in or out of the classroom, who wants to learn to program the Commodore 64 computer.

This book is structured to make learning easy. Each chapter begins with a statement of objectives. Then discovery exercises let the student experiment with BASIC and see the language in action. Once students acquire a feel for BASIC, they can profitably proceed to a more traditional treatment of the concepts.

The text has an introduction, 12 chapters and three appendices. Each chapter is a module of instruction that should require

about one or two hours of computer work and perhaps one or two hours text study. Reviews at the end of each chapter let students test their mastery of the objectives. The book can be used in different ways: as a self-study text; as the text for an open-entry, open-exit, self-paced course; and in tandem with a traditional lecture course.

People at any level from junior high through graduate school should be able to use this book to learn programming skills in BASIC rapidly and effectively. The student needs no knowledge of mathematics past introductory algebra, and the algebra used is mainly formula evaluation. Students with more advanced mathematical skills can apply them to independent work on the computer.

All students will need access to a Commodore 64 computer, at least one disk drive, and a black-and-white or color TV set. Although most of the work in the book can be done without a disk drive, the lack of a disk drive significantly limits the potential of the Commodore 64 computer.

The documents furnished with the Commodore 64 and the VIC-1541 Disk Drive are valuable to the student. The *Commodore 64 User's Guide* tells how to connect the Commodore 64 to a TV set and has all the specifications and capabilities of Commodore 64 BASIC. The *VIC-1541 Single Drive Floppy Disk User's Manual* describes the use of a disk drive with the Commodore 64.

### Acknowledgments

I thank Wade Ellis, Jr. and Ed Lodi of the Computer Tutors of San Jose, California, for their assistance in writing this adaptation of *BASIC: A Hands-on Method* and for their typesetting and composition of the book using the T<sub>E</sub>X composition system. Thanks, too, to Antonio Padial and Loralee Windsor for their excellent editorial consultation.

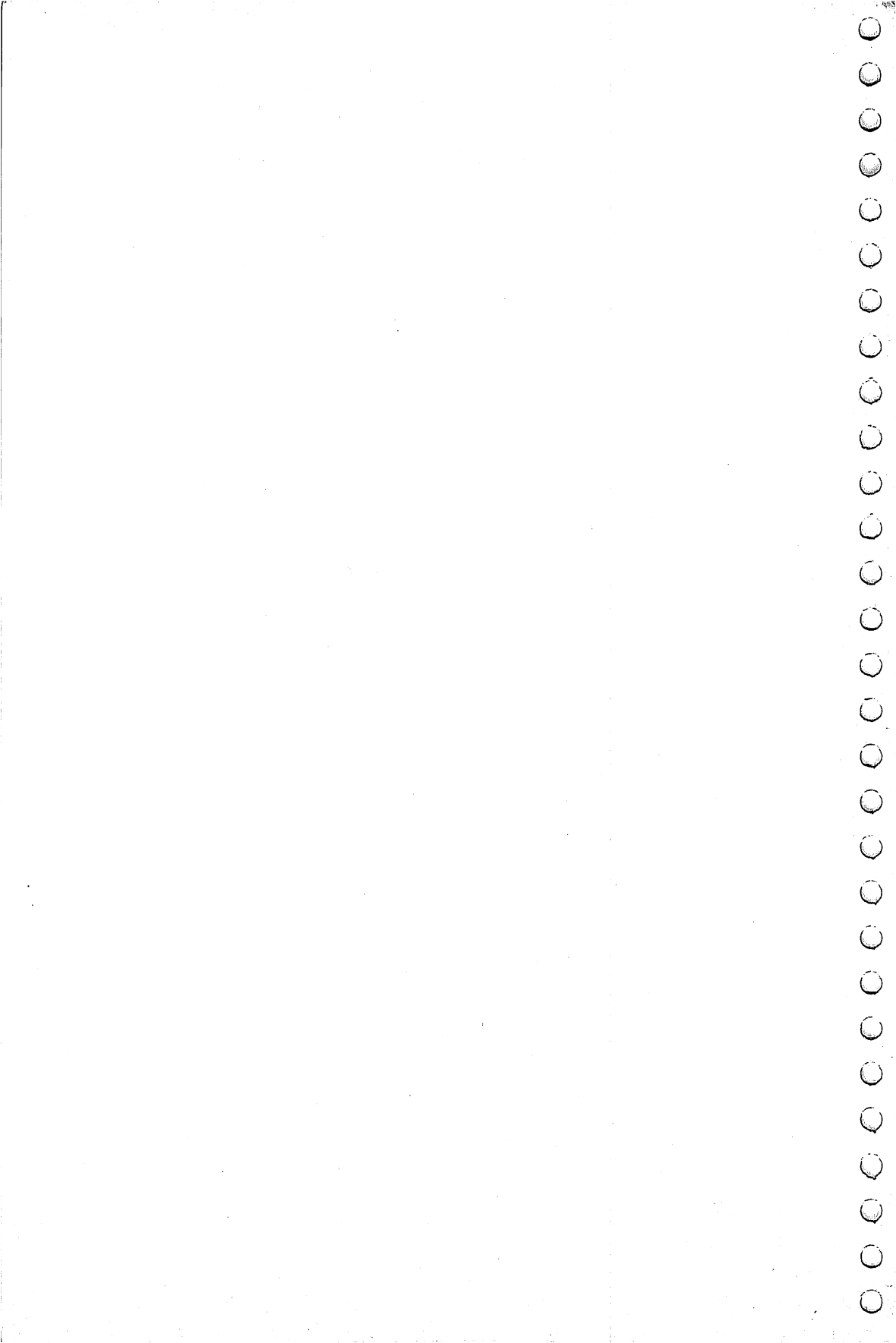
Herbert D. Peckham

We would like to thank Jane Ellis for working through the computer exercises and the practice test solutions. The staff of the Context Project at Stanford University were, as always, extremely helpful. Special thanks are due to Dikran Karagueuzian who helped us

immensely with transferring files and for his efforts in getting the files to their final form.

Comments or suggestions for improvement of this book will be appreciated.

Wade Ellis, Jr.  
Ed Lodi



# INTRODUCTION

Computers play a part in most of our daily activities. Without computers, businesses of all sizes, educational institutions, and various branches of government would be unable to handle the bewildering quantity of information that characterizes our society. Although the routine use of computers has become a significant part of everyday activities only in recent years, the trend will surely continue as the price of computers continues to drop. More and more people will need to know how to use computers if they are to participate fully in our society.

## **What Is BASIC?**

You are about to embark on the study of the computer language called BASIC. BASIC is a very specialized language that permits communication between you and the computer. This language is not complicated and is certainly much easier to learn than Spanish or French. BASIC has a simple vocabulary of a few words, a grammatical structure, and rules of use just like any other language. Your main tasks will be to learn the vocabulary of BASIC, become familiar with its rules of grammar, and begin to see how the language lets you use the computer to do what you want. We have intentionally kept the level of mathematics in this book simple. Don't be too concerned if your mathematical skills are a bit rusty. As we proceed, you will have an opportunity to brush up on elementary mathematics.

One effective way to learn is to observe details and characteristics while performing a task. We will use this "discovery" strategy. You will begin each chapter with a session on the computer. After following the directions and observing the computer's response to your instructions, you will begin to acquire a feel for BASIC. Then



you will study written material that summarizes what you have learned.

### **Where Did BASIC Originate?**

The original version of BASIC was designed and written at Dartmouth College under the direction of Professors John G. Kemeny and Thomas E. Kurtz. In September 1963 they began to create a programming language written from the user's point of view. Much of the actual programming on the project was done by Dartmouth undergraduate students. Professors Kemeny and Kurtz introduced the completed language on May 1, 1964.

The success of this pioneering effort at Dartmouth attracted national attention, and other institutions became interested. The rest is history. What started as a project at a single college is now an established part of the computer industry. Today nearly every time-sharing computer "speaks" some version of BASIC.

The enhanced versions have significantly increased the power and capability of the language. The most recent development is the adaptation of BASIC for use on small, inexpensive personal computers. Commodore 64 BASIC, the language presented in this book, is a powerful and flexible enhanced version of BASIC.

### **How to Use This Book**

Each chapter begins with a brief statement of the objectives. Study these objectives carefully to get a clear picture of precisely where you're going.

The next section of each chapter contains the discovery exercises. In that section you will record the computer output in the space provided, when appropriate, and try to answer any questions you are asked. These activities will lead you through the ideas involved and let you see BASIC working. Try to think about what will happen in the situations that are set up.

Your relationship with the computer should be an active one. Whether your answers are correct is not important. The important thing is to think carefully about the questions and try to answer them. The time you spend thinking about questions will save you time later on.

Following the discovery exercises is a complete discussion of the objectives. Since you will already have seen the ideas and

concepts in action on the computer, your study of this material will be much easier and more profitable.

Typical programs are included in each chapter. These are discussed in detail to show how elements of programming are pulled together to produce a BASIC program.

Beginning with Chapter 4, we give a set of problems at the end of each chapter. You should work enough problems to satisfy yourself that you can write programs at the appropriate level. Solutions to the odd-numbered problems are in Appendix E.

Finally, each chapter has a practice test that lets you review the material and discover needs for further study. The answers to the practice tests are contained in Appendix D.



# GETTING ACQUAINTED WITH YOUR COMMODORE 64 COMPUTER

Since the computer may seem a bit strange and complicated at first, we will proceed slowly. After a few sessions, routine operations will seem very natural and will cause you no trouble. Initially, though, be prepared for a certain "confusion quotient." Don't hesitate to review previously studied material if necessary.

## 1-1 OBJECTIVES

In this chapter you will become familiar with the computer and start learning how it operates. You will do no BASIC programming until the next chapter. Learning how to operate the keyboard and how to enter and modify information is easy, but it is fundamental to all that follows.

### **Learning to Start the Computer**

You will learn some essential preliminaries, including how to turn the computer on and off and how to set the TV antenna switch and channel selector.

### **Using Direct Mode**

One of the easiest ways to use the computer is in the direct mode. No programming is involved; the computer carries out instructions as they are entered. In due time you will learn how to do much more. For the present, however, simple operations in the direct mode are a good introduction to using the computer.

## Correcting Mistakes

The information entered into a computer is rarely mistake free. You need to know how to edit— how to change or correct material that has been entered.

## 1—2 DISCOVERY EXERCISES

Before beginning work on the computer, we must establish several important points. On a typewriter, the letter L is often used for the numeral 1. On the computer, however, the numeral 1 is with the other numeral keys along the top row of the keyboard. Similarly, the letter O is sometimes used for the numeral 0 on a typewriter, but on the computer the 0 is found on the top row of the keyboard.

- **Don't use the L for the 1.**  
**Don't use the O for the 0.**

The plus sign (+) is typed without holding down the shift key. The symbols —, \*, and ↑ are also typed without using the SHIFT key.

The disk drive will not be used until Chapter 3. You need not install it or turn it on until then.

The antenna switchbox should be connected to the TV set. A cable should be connected to both the antenna switchbox and the computer. The TV channel selector switch should be set to channel 3 or 4 (depending on whether the small white CHANNEL SELECTOR switch on the back of the Commodore 64 is set to left or right). Instructions for connecting the TV set to the computer can be found in the *Commodore 64 User's Guide*.

1. Now we are ready to begin work. Sit down in front of the computer, get comfortable, and locate the key marked **RETURN** at the right side of the keyboard. This key will be referred to as RETURN ; you will use it often.
2. Be sure the antenna switch is set to **COMPUTER** and turn on the TV set with the correct channel setting. Make sure there is no program or game cartridge in the slot in the back of the Commodore 64. Now turn on the computer with the ON/OFF switch recessed on the right side of the cabinet. A red light on the right of the cabinet will come on to indicate that the computer is on. As soon as your TV set warms up, you will see the message



```
**** COMMODORE BASIC V2 ****  
64K RAM SYSTEM 38911 BASIC BYTES FREE
```

```
READY.  
■
```

or something similar. You may wish to adjust the TV set to get a clearer picture. If you do not see these messages, start the procedure over again. The blinking square is called a cursor. It will be displayed on the screen most of the time.

3. Now type

```
PRINT 1+4
```

If you get only lower case letters, i.e., **print 1+4**, press the SHIFT key and the Commodore key **C=** simultaneously and continue. Do not hold down the SHIFT key when you type +. Has anything happened?

---

Now press **RETURN** and record what happened.

---

4. Now you know how to make the computer do addition. Let's explore this further. Type

```
PRINT 20+54.7
```

and press **RETURN** . What happened?

---

5. Type

```
PRINT 2+4-3
```

and press **RETURN** . Record the output below.

---

## 6. Type

```
PRINT 12/2
```

and press `RETURN` . What happened?

---

What arithmetic operation does the / call for?

---

7. If you make a typing error, you can move the cursor, `■` , back to the error by pressing the key marked INST/DEL at the upper right of the dark brown keys. We will refer to the key as the delete key. Each time you press the delete key, the cursor will move one place to the left, erasing a character. When you reach the error, retype the line correctly. Sometimes, after you press `RETURN` , you may see the message `?SYNTAX ERROR`. If this happens, try to see what the problem is and retype the entire line correctly.
8. Your TV screen should be fairly full now. Hold down the SHIFT key and press the key marked CLR/HOME next to the delete key at the right side of the keyboard. We shall call this operation `SHIFT|CLR/HOME` . What happened?
- 

Pressing `SHIFT|CLR/HOME` clears the screen. If the cursor is not on the screen, press `RETURN` several times. If the screen is full and new lines are entered, old lines will scroll off the top.

## 9. Type

```
PRINT 2*50
```

and press `RETURN` . Again, do not hold down the SHIFT key when typing the \*. What happened?

---

What arithmetic operation is called for by the \*?

---

10. Type

```
PRINT (2+3)*4-1
```

but don't press  . What do you think will happen when you press  ?

---

Press  and record what happened.

---

11. Type

```
PRINT "(2+3)*4-1"
```

and press  . What happened?

---

12. What will happen if you type

```
PRINT "BAD DOG"
```

and press  ?

---

Try it and see if you were correct.

13. Let's move on to a different topic. First, clear the screen. If you have forgotten how, look back at step 8. Type

```
GRADE=95
```

and press  . Then type

```
PRINT GRADE
```

and press  . What happened?

---

Now type

```
?GRADE
```

and press **RETURN** . What happened?

---

The question mark is an abbreviation for PRINT.

14. Take a few moments to examine the lines below.

```
WIDTH=10  
DEPTH=6  
HEIGHT=4  
VOLUME=WIDTH*DEPTH*HEIGHT  
PRINT VOLUME
```

What do you think will happen if you type in these lines?

---

Now type in the lines. Remember to press **RETURN** at the end of each line. What happened?

---

15. Study the lines below briefly.

```
WIDTH=12  
DEPTH=9  
SQYDS=(WIDTH*DEPTH)/9  
PRINT SQYDS "SQYDS"
```

What will happen if you type these lines?

---

Clear the screen and type the lines, but remember to press **RETURN** after each line. What happened?

---

16. Now go on to a different topic. Clear the screen. Press the SHIFT LOCK key on the left of the keyboard. Now press the keys marked A, S, D, F, G, H, J, K, and L. What happened?
- 

Pressing **SHIFT LOCK** places the computer keyboard in a graphics mode. Look at the fronts of the keys you have pressed to see the graphic character generated in this mode. Not all keys generate graphics characters. You may wish to experiment with the other keys on the keyboard.

17. Now hold down the SHIFT key on the left and press the key marked **C=** just to the left of the SHIFT key. What happened?
- 

Repeat this operation several times observing the screen as you do so.

18. This concludes the discovery material for this chapter. Turn off your computer and TV set.

### 1-3 DISCUSSION

#### Learning to Start the Computer

It is very simple to turn the computer on and off. As you have already seen, you use the ON/OFF switch at the right side of the computer cabinet. Before turning on the computer, be sure to remove any program or game cartridge installed in the slot in the back of the cabinet.

**WARNING: NEVER REMOVE OR INSERT A PROGRAM CARTRIDGE OR GAME CARTRIDGE WHILE THE COMMODORE 64 IS ON.**

You must turn on the TV set with the antenna switch set to **COMPUTER** and the channel selector switch set to 3 or 4. After you have done these things, you will see a start-up message and

READY.



will appear on the screen.

The computer starts up in upper case/graphics mode. You may shift into (and out of) the upper/lower case mode by pressing the **C=** and **SHIFT** keys simultaneously. You will use the upper case/graphics mode throughout this book.

One important point: If things get away from you, if you lose touch, or if the computer seems out of control, you have a way to regain control. Simply turn off the computer with the **ON/OFF** switch recessed in the right side of the cabinet. The disadvantage to this remedy is that you will lose programs or information in memory when you recover in this manner. Alternatively, you may hold down the **RUN/STOP** key and press the **RESTORE** key. This action erases the direct mode statements on the screen, but does not erase the information stored in memory.

### Using Direct Mode

In the discovery activities, you learned how to use the computer like a simple calculator to do simple arithmetic operations. This is also known as the direct mode. As you shall see in the next chapter, you can use BASIC to store statements and commands in a series of numbered lines and then direct the computer to perform all the statements. If, however, you type in the statements without line numbers, the computer assumes you want a direct or immediate answer and does what you asked it to do, if possible.

When you type in material, nothing happens until you press **RETURN**. The **RETURN** key tells the computer you are through typing a piece of information. In a few cases, the computer responds to a single keystroke and you do not have to press **RETURN**. Such cases, however, are the exception rather than the rule.

You have discovered that addition and subtraction are called for by **+** and **-**, which probably wasn't much of a surprise. Multiplication and division are indicated by **\*** and **/**, respectively. All these mathematical operators are typed without pressing the **SHIFT** key. Parentheses can be used to group operations any way you wish. A number of other clever operations are possible, but we will postpone discussion of these until later chapters. If you type

```
PRINT 5*3.2+6.3
```

and press **RETURN**, the computer will carry out the arithmetic and print the result.

If you type

```
PRINT "ABCDEFGG"
```

and press **RETURN**, the computer prints out the collection of characters between the quotation marks— in this case the letters ABCDEFG. Such a collection is called a “character string,” an important concept that we will return to throughout the book.

The computer can keep track of a number of pieces of information in the direct mode. Thus, if you type

```
A=2  
B=3  
PRINT A+B
```

the computer will print 5 on the screen. There is a very important point in connection with this concept. If you type

```
PRINT TAX
```

and press **RETURN**, the computer will display 0. Since you gave no value to TAX, the computer assigned the value 0 and printed it.

The computer is very relaxed about names for quantities used either in the direct mode or in BASIC programs. You can use long names like DEPTH or RATE as well as short names like D or R. However, you need to be careful when using long names for this reason: The computer uses only the first two characters to distinguish between variables. Thus, the computer interprets BAR and BAT as the same variable in Commodore 64 BASIC. Also, spaces should not be included in names. Certain words cannot be used for variable names, because they are reserved for use by the computer. The list of reserve words appears in Appendix D of the *Commodore 64 User's Guide* under the heading COMMAND.

### Correcting Mistakes

Later, you will learn to use Commodore 64 BASIC line editing commands to make changes. These commands are most effective when used to modify BASIC programs. However, you can edit a line in the direct mode as long as you have not yet pressed the **RETURN** key to end that line. You edit the line by moving the cursor to the left with the delete key. Each time you press the delete key you erase

a character to the left of the cursor. After you correct the error, you can resume typing from that point.

### 1-4 PRACTICE TEST

Take the test below to discover how well you have learned the objectives of Chapter 1. The answers to this and all subsequent practice tests are given in Appendix D.

1. How do you let the computer know that you are through typing a line?

---

2. If you lose control of the computer, how can you regain it?

---

3. What symbol indicates multiplication on the computer?

---

4. How do you clear the screen display?

---

5. What operation does the symbol / indicate?

---

6. What will happen if you type

`PRINT 3*4/6`

and then press `RETURN` ?

---



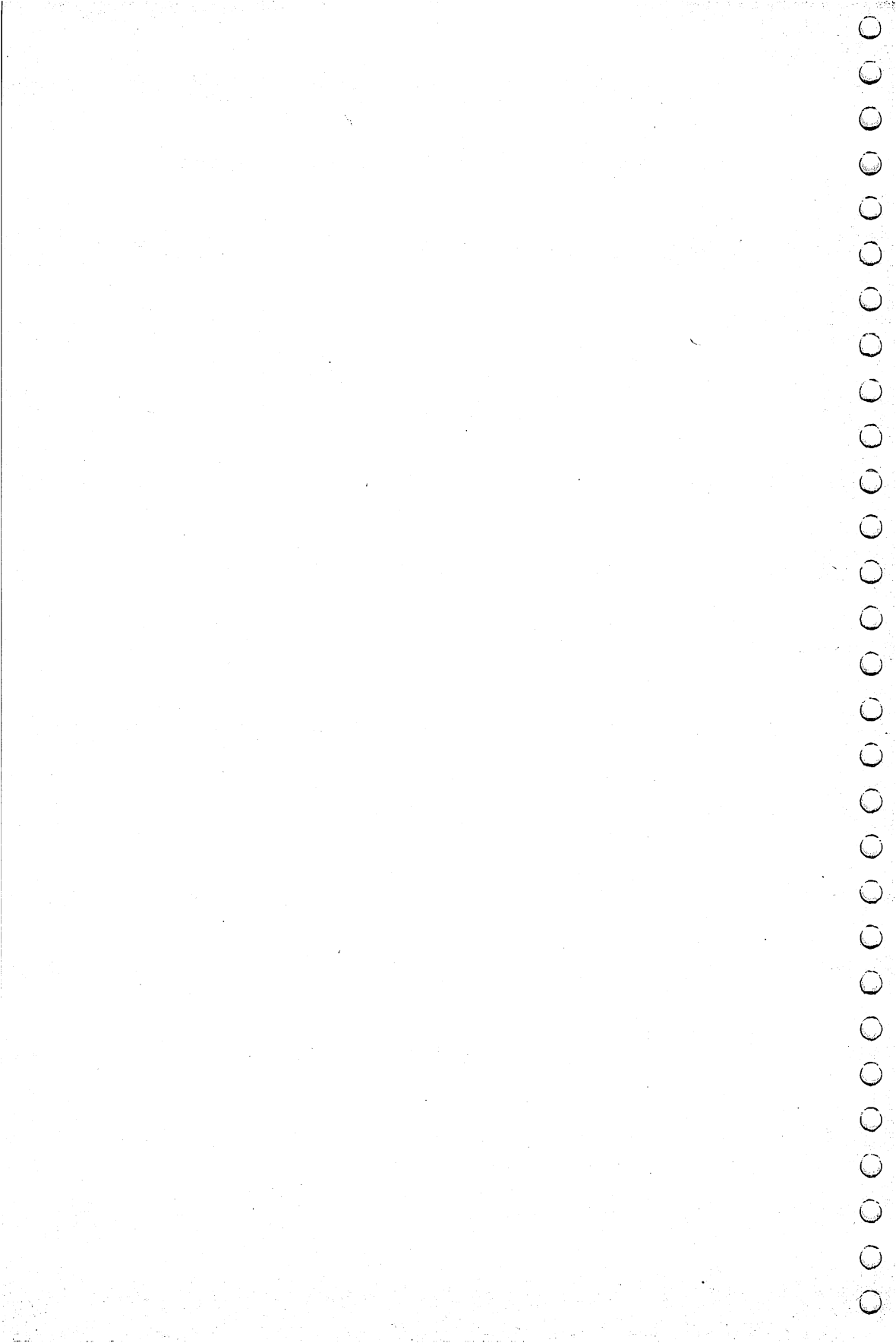
7. What will happen if you type

`PRINT "25/5+2"`

and then press `RETURN` ?

---

8. If you typed `PRING 2+3*4` and noticed your spelling error before you pressed `RETURN` , how could you correct the error?
-



# INTRODUCTION TO BASIC

Now you are ready to begin learning about programming in BASIC. In this chapter you will see how to write and run some very simple programs.

## 2-1 OBJECTIVES

### **Learning to Edit Programs**

You already know how to correct a line using the delete key. You will learn to replace a line in a program simply by typing a new line with the same number, and you will learn to use the editing features of Commodore 64 BASIC to add information to or delete information from a line. A thorough knowledge of editing will save you time later on.

### **Learning the Requirements for BASIC Programs**

All BASIC programs have common characteristics. You will look at some very simple programs to learn about these characteristics.

### **Telling the Computer What to Do**

Commands are action words that tell the computer to do something to or with a BASIC program. These action words are used to control a program. You will look at the following commands: LIST, RUN, and NEW.

### **Entering and Controlling Programs**

This objective overlaps the one above. The main thing is to get sufficient exposure so that you will feel comfortable entering and

controlling programs. All the programs you will encounter initially are short and easy to handle.

### Assigning Variable Names in BASIC

You must know how to name either numbers or strings of characters in BASIC programs. Fortunately, the computer's rules for assigning names are very relaxed.

## 2-2 DISCOVERY EXERCISES

In the discovery activities that follow, you will be directed to enter various programs. If you see a `<RETURN>` in the instructions, press `RETURN`. Remember from your experiences in Chapter 1 that pressing `RETURN` tells the computer you are through typing. Now go on to the activities below.

1. Turn on the TV set and your computer. Be sure the antenna switch and TV channel selector switch are set correctly.
2. Type in

```
100 LET A=1 <RETURN>
```

This is the first line of a BASIC program.

3. Now type in the balance of the program as listed below.

```
110 LET B=8 <RETURN>  
120 LET C=A+B <RETURN>  
130 PRINT C <RETURN>  
140 END <RETURN>
```

If you make mistakes before you finish typing a line, correct it using the methods learned in Chapter 1. If you notice the mistake in a line after you have finished entering it, do not try to return to that line. Simply retype it at any point in the program. The new line will supersede the old one.

4. Clear the screen using `SHIFT|CLR/HOME`. What happened to the program you just typed in?

---

5. Fortunately, all is not lost. The computer remembers what you typed in even though the screen is blank. Type LIST and press **RETURN**. What happened?
- 

6. On the TV display you should see the program you just entered. For the time being ignore the line numbers at the beginning of each line. Just read the lines in the program and try to get a sense of what they mean. If the computer is told to carry out the instructions, what do you think will happen?
- 

Type the letters RUN and press **RETURN**. What happened?

---

7. Now type

```
110 LET B=5 <RETURN>
```

Clear the screen, type LIST, and then press **RETURN**. Is the last number in line 110 an 8 or a 5?

---

8. If you tell the computer to execute this program what do you think will happen?
- 

Type RUN, press **RETURN**, and record what happened. Were you right?

---

9. Now type

140 <RETURN>

clear the screen, and display the program using the LIST command. What has happened to line 140?

---

If you want to delete a line in a BASIC program, how do you do it?

---

10. Now run the program (type RUN and press RETURN ). What happened?
- 

Does the computer appear to require the END statement that formerly was in line 140?

---

11. Experiment a bit more. Often you will want to clear out the program in the computer's memory. You use the NEW command to clear the memory. Type NEW and press RETURN . What happened?
- 

Type LIST and press RETURN to see what the computer has in memory. Is anything there?

---

12. You have learned how to clear out a program in memory, but now you have no program left! To get the program back you must enter it again. Type in the program below.

```
100 LET A=1 <RETURN>
110 LET B=8 <RETURN>
120 LET C=A+B <RETURN>
130 PRINT C <RETURN>
140 END <RETURN>
```

Check all the lines to make sure you entered them correctly. If a line needs to be changed, retype it. If you had to retype lines, clear the screen with SHIFT|CLR/HOME and redisplay the program by typing LIST.

13. Now type

```
125 LET D=B-A <RETURN>
135 PRINT D <RETURN>
```

Clear the screen and display the program. What has happened?

---

14. Take a few moments to study the program. What will happen if you run the program?
- 

Type RUN, press RETURN , and record below what the computer did.

---

15. In the original program the line numbers were not consecutive (line 100, 101, 102, 103, etc.) but had gaps (e.g., 100, 110, 120, 130, and 140). Can you think of a reason for leaving these gaps? (Hint: See step 13.)
- 

16. How do you insert lines in a BASIC program? (Hint: See steps 13 and 15.)
-

17. Clear the program in memory by typing NEW and pressing **RETURN**. Enter the program below.

```
100 INPUT WHITE <RETURN>
110 LET RED=WHITE+2 <RETURN>
120 PRINT RED <RETURN>
130 GOTO 100 <RETURN>
140 END <RETURN>
```

18. This new program has several features you have not seen before. Study the program carefully and think about what will happen if you run the program. What does GOTO 100 in line 130 mean?

---

19. Run the program and record what the computer did.

---

Type the numeral 6 and press **RETURN**. What happened?

---

20. Type the numeral 10 and press **RETURN**. What took place?

---

21. What line in the program do you think is generating the question mark?

---

Describe in your own words what the program is doing. If necessary, experiment some more to make sure you are correct.

---

22. Now get out of the program at the question mark. Locate the keys marked RUN/STOP and RESTORE on the far left and right of the keyboard. Hold down **RUN/STOP** and press **RESTORE**. From now on we will refer to this as **RUN/STOP|RESTORE**. What happened?

---



If nothing happened, press **RUN/STOP|RESTORE** again and continue.

23. Use NEW to clear the program in memory. Type in the following program.

```
100 LET A=100 <RETURN>
110 PRINT A <RETURN>
120 LET A=A+1 <RETURN>
130 GOTO 110 <RETURN>
140 END <RETURN>
```

Run the program and record below what happened.

---

When you get tired of watching the display, press **RUN/STOP**. What happened?

---

24. Try it once more. Run the program and after a few numbers are typed out, interrupt the program. How do you stop a BASIC program running on the computer?

---

25. Clear the screen and display the program in memory. (See steps 4 and 5.) Type the lines below. Note the absence of spaces in the first line and the extra spaces in the second.

```
100LETA=1 <RETURN>
1 2 0 L E T A = A + 1 <RETURN>
```

Clear the screen and list the program. Are lines 100 and 120 displayed as you typed them?

---

Run the program. What happened?

---

Press **RUN/STOP** to interrupt the program. Clearly, some spaces are important in BASIC statements, others are not. Just note the fact for now. We will return to this matter later.

26. Let's try a program with some new features. Clear the program from memory by typing NEW and then pressing **RETURN**. Type in the program below.

```
100 PRINT "TYPE A NUMBER" <RETURN>
110 INPUT FIRST <RETURN>
120 PRINT "ONE MORE TIME" <RETURN>
130 INPUT LAST <RETURN>
140 LET SUM=FIRST + LAST <RETURN>
150 PRINT "THEIR SUM IS" <RETURN>
160 PRINT SUM <RETURN>
170 END <RETURN>
```

27. Study the program for a few moments. Now RUN the program. What happened?

---

Type the numeral 12, press **RETURN**, and record below what the computer did.

---

28. All right, now type the numeral 13, press **RETURN**, and record below what happened.

---

29. We would like to change ONE MORE TIME to ONE MORE NUMBER in line 120. Of course, we could retype the entire line, but instead we will use the editing features of Commodore 64 BASIC. Type

```
LIST 120 <RETURN>
```

What happened?

---

Hold down the SHIFT key and press the key marked **↑CRSR↓** at the lower right of the keyboard. What happened?

---

Press **SHIFT|↑CRSR↓** two more times. The cursor should now be on line 120. Release the SHIFT key and press the key marked **←CRSR→** several times until the cursor is on the second quotation mark. Press the delete key 4 times to delete the letters E, M, I, and T to the left of the cursor. Now type NUMBER" and press **RETURN**. What happened?

---

List the program. Line 120 should look like

```
120 PRINT "ONE MORE NUMBER"
```

30. Display the program to see that the change has been made. If the line is not correct, you may wish to retype it or use the editing features to correct it.
  31. A line may be changed by listing it, moving the cursor (abbreviated CSRS) to the position where changes should be made and then typing the changes. Pressing **RETURN** places the changed line in the program.
  32. Let's make another change in line 120. Display the line. Move the cursor to the second quote mark at the end of the line. Hold down the SHIFT key and press INST/DEL 1 time. What happened?
- 

33. Now press **SHIFT|INST/DEL** 7 more times. You **must** now insert 8 characters (or press **RUN/STOP** to end the editing of the line). The delete key and cursor keys will not work properly now. Type

```
, PLEASE
```

What happened?

---

34. Now press **RETURN** to complete the editing of the line. Display the program to see that the changes have been made. Line 120 should look like

```
120 PRINT "ONE MORE NUMBER, PLEASE"
```

35. Run the program. At the input prompts, type 22 and 23. Is the program more polite now?
- 

36. Now look at a different topic. Clear the screen. Type NEW and press **RETURN** to clear the program in memory. Then enter the following program.

```
100 LET A=1 <RETURN>
110 LET A$="HOUSE" <RETURN>
120 PRINT A <RETURN>
130 PRINT "A" <RETURN>
140 PRINT A$ <RETURN>
150 PRINT "A$" <RETURN>
160 END <RETURN>
```

37. This program contains something new. Look at the A\$ in line 110. Note that it is set equal to a word enclosed in quotation marks. The balance of the program has to do with variations on printing out A and A\$. Run the program and record the output.
- 

38. Study the output carefully and identify what was printed in response to each of the PRINT statements. For the time being just make the comparison. Later we will examine the subject in detail. Enter the following line:

```
155 PRINT B <RETURN>
```

39. Clear the screen and display the program with the LIST command. Note that B is mentioned only in the PRINT statement in line 155. What do you think will happen if we RUN the program?
-

All right, now run the program and record what happened.

---

40. As you saw in Chapter 1, even though the value of B was not defined in the program, the computer assigned it a value of 0. This is an important fact to remember while writing programs.
41. This concludes the discovery activities for this chapter. Turn off the computer and the TV set and go on to the next section.

## 2-3 DISCUSSION

### Learning to Edit Programs

Since most of us make mistakes while typing, we need to be able to correct errors. Suppose you make a mistake while you are typing a line. If you have not yet pressed **RETURN** to end that line, you can move the cursor left using the delete key and make corrections as you saw in Chapter 1. When all the corrections are made in a line, press **RETURN**. If you wish to change a line after you have finished typing it (after you have pressed **RETURN**), you may retype the line or you may use the editing features of Commodore 64 BASIC. For example, if you wish to change line 110 in a program, you should type in

```
LIST 110 <RETURN>
```

The computer will display line 110. Use the cursor keys at the lower right of the keyboard to place the cursor where the changes are to be made. Type over existing characters or use the insert/delete **INST/DEL** key or the **SHIFT|INST/DEL** key to make changes. Press **RETURN** to end the editing of the line. Spaces inserted with the **SHIFT|INST/DEL** key must be deleted (with the delete key) before other characters can be deleted. After **SHIFT|INST/DEL** has been pressed once, if **INST/DEL** is pressed, a reverse T will appear. Pressing **INST/DEL** again will delete this character.

### Learning the Requirements For BASIC Programs

You have seen several important facts about BASIC programs demonstrated. As a point of reference, we will use a program you

used in the computer work:

```
100 LET A=1
110 LET B=8
120 LET C=A+B
130 PRINT C
140 END
```

Each BASIC program consists of a group of lines called statements. Each statement must have a line number and be less than two lines long (80 characters). In the program above, there are three types of BASIC statements: LET, PRINT, and END. The first two statements will be treated fully in the next chapter. For the time being the use of each of these statements in the program is clear. The Commodore 64 will execute a program with or without the END statement. However, it is a good practice to use it, especially in long programs, so there will be no confusion about where the program ends. We will make it a rule to use the END statement in all programs even though the computer doesn't require it.

Generally, the line numbers in a BASIC program are not numbered consecutively (such as 100, 101, 102, etc.). The reason is that often programmers need to insert additional statements later if they discover errors or want to modify the program. If the lines were numbered consecutively (spaced 1 apart), it would be less convenient to make changes. When there are gaps in the line numbers, statements can be inserted simply by typing in the new statements and assigning them line numbers not already in the program.

As you saw in the Discovery Exercises, some spaces are important in BASIC programs. Keep improper spacing at the back of your mind as a possible reason for error messages. If you make your program statements readable, you should have no problems with spaces.

You can enter program lines in any order. If, for example, you type

```
140 END
120 LET C=A+B
110 LET B=8
130 PRINT C
100 LET A=1
```

and list this new program, the computer will sort out the statements

and display them in numerical order. In the same way, if you ran the program as typed above, the computer would first arrange the statements into numerical order before it executed them.

You can remove a BASIC statement from a program by typing the line number and pressing RETURN. You can modify statements by retyping the lines involved, pressing RETURN after each line is typed, or by using the editing features. As indicated above, you can add statements by using line numbers not already in the program. Thus, you can add, remove, or change BASIC statements as you wish. This ability to change programs easily is one of the powerful characteristics of Commodore 64 BASIC.

### **Telling the Computer What to Do**

There is an important distinction between the statements in a BASIC program and commands. Commands tell the computer to do something with a program. You have seen several commands in the computer work, and we will briefly review the use of each.

When you enter a BASIC program, it goes into memory. Quite often you need to see the program contained in memory, perhaps because you need to make changes in the program, or because you simply need a copy of the program. In any case, you already know you can use the LIST command to have the computer display the program.

The LIST command is very useful. If a program doesn't work as it should, your first step should be to display the program in memory. Use the displayed copy to troubleshoot the program.

When you turn off the computer, the contents of memory are cleared out automatically. While working on the computer, however, you may unknowingly mix programs together. Suppose you are working with one program and decide to go on to another. If you don't clear the first program out of memory, the second program will go in to memory right over the first. As a result, parts of both programs may be in memory. The way to avoid jumbling programs is to clear (or erase) a program with the NEW command when you are through with it.

A BASIC program is simply a set of instructions on which the computer will act. However, the computer needs to be told to start this process. When the computer receives the RUN command, it goes to the lowest numbered statement in the program, carries out the instructions, goes to the next higher numbered statement,

and keeps on carrying out instructions in numerical order, unless the program directs that a statement be done out of order.

- Use **SHIFT|CLR/HOME** to clear the screen. Then type LIST to display the program in memory.
- Type NEW to clear the program in memory.
- Type RUN to have the computer execute a program.

### Entering and Controlling Programs

So far, in the instructions for typing programs or commands, the **(RETURN)** prompt reminded you to press the RETURN key. This habit should be well developed now, so we will not use the **(RETURN)** prompt in the future. From now on, when you are through typing a statement or command, press **RETURN** to let the computer know you are finished.

At times, you will need to control a program that is running. An extreme case is a program in a closed loop. Such a program will keep on running forever if you don't interrupt it. You can break into a program by pressing **RUN/STOP**. When you press **RUN/STOP** the computer breaks the program execution and displays the words, **BREAK IN** followed by the number of the line that was being processed when the interruption took place. A different situation occurs when the computer is in an input loop waiting for a number to be typed in. If you want to get out of an input loop, press **RUN/STOP|RESTORE**. The computer then jumps out of the program execution back to the **READY** mode.

### Assigning Variable Names in BASIC

One area of confusion for beginners is the distinction between the name of a variable and the data stored under that name. In the BASIC statement

```
100 LET A=2
```

the **A** names a variable, which means that different values can be assigned to **A**. Consequently, **LET** statements are often called assignment statements. In this case, the variable **A** is assigned the value 2. Actually, what is taking place is that somewhere in the computer



there is a memory location named **A**, and the number 2 is stored in that location. The fundamental idea is to separate the name of a location in memory from the contents of the memory location. The difference is the same as that between a post office box number and the contents of that box. The box number does not change, but the contents of the box may change at any time.

Consider the following statement.

```
130 LET C=A+B
```

This statement instructs the computer to get the numbers stored in locations named **A** and **B**, add them together, and put the sum in the storage location named **C**. The equal sign tells the computer to evaluate what is on the right and assign it to the variable name on the left.

Now consider the BASIC statement:

```
120 LET B=B+1
```

Suppose you consider the statement as an algebraic equation:

$$B = B + 1$$

If you subtract **B** from both sides of this equation, you get

$$0 = 1$$

which is very strange indeed! It is certainly clear that in a BASIC statement, the equal sign does not mean what it does in an algebraic equation. Instead, the statement

```
120 LET B=B+1
```

instructs the computer to get the number stored in location **B**, add 1 to the number, and put the result back in the storage location named **B**.

Once you store a number in a location, anything that was stored there before is lost. Consider the following statements:

```
100 LET A=1  
110 LET A=2*3
```

Line 100 instructs the computer to set up a location called **A** and put the number 1 in that location. Line 110 tells the computer to multiply 2 by 3 and store the product in location **A**. The 1 stored previously in location **A** is lost.

This brings us to the heart of the issue. The letter **A**, which identifies a location, is called a variable, because the contents of **A** can be changed. The name of the location does not change, but the number stored there can change.

To be precise, the variable **A** above is a numeric variable. The reason for including numeric in its name is that another type of variable exists—the character string variable. You were introduced to this concept briefly in the discovery activities.

It is easy to distinguish between numeric and character-string variables. **A**, **BOOK**, **M**, and **P** all identify numeric variables and name numeric quantities. **A\$**, **BOOK\$**, **M\$**, and **P\$** all name strings of characters. The **\$** symbol identifies the name as a character-string variable. In the BASIC statement

```
100 LET B$="BARN"
```

**B\$** names a location in memory in which the character string "BARN" is stored. The quotation marks set off the string, but are not part of it.

Recall that Commodore 64 BASIC has very relaxed rules for variable names. It allows you to use "long" names for numeric variables as well as character strings. You can use up to 79 characters (including the **\$** character in the case of character strings) in long names. However, the computer distinguishes among variables only by the first two characters of the name. Also, the computer has a set of words "reserved" for use in BASIC and for commands. These words cannot be used to name variables. If you make a mistake and use one, however, the computer will let you know!

To summarize, a variable name in BASIC identifies a storage location in memory. That location can contain a number or a character string. The contents of the storage location (the value of the variable) can be changed, but the name of the location cannot.

The **LET** (or assignment) statement tells the computer to evaluate what is on the right side of the equal sign and assign it to the storage location named on the left side. Thus,

```
100 LET D=A+B+C
```

instructs the computer to use the numbers stored in memory locations named A, B, and C to evaluate the expression  $A+B+C$  and to store the result in the memory location named D.

The use of LET in the assignment statement is optional in Commodore 64 BASIC. For the sake of consistency, in this book we will always use LET in assignment statements.

We have just scratched the surface with regard to character-string variables. We will return to this topic several times during the balance of the book.

## 2-4 PRACTICE TEST

1. How do you signal the computer that you are through typing a line or instruction?  

---

2. Suppose that your computer is waiting at an INPUT statement in a program for you to enter a number. You decide instead that you want to jump the computer out of the program. How do you do this?  

---

3. How do you interrupt a program that is running on your computer?  

---

4. What is wrong with the following program?

```
100 LET A=1
110 LET B=3
120 LET C=B-A
PRINT C
130 END
```

---

5. What would happen if you corrected and ran the program in question 4?

---

6. How long can long variable names be?

---

7. How do you insert a line in a BASIC program?

---

8. How do you replace a line in a BASIC program?

---

9. How do you remove a line from a BASIC program?

---

10. How do you display the program in memory?

---

11. How do you erase the screen?

---

12. How do you command the computer to execute the program in memory?

---

13. How do you erase a program in memory?

---

14. What is the difference between a numeric and a character-string variable?

---

15. What will the following editing command do?

`LIST 120`

---

16. What keys are used to move the cursor?

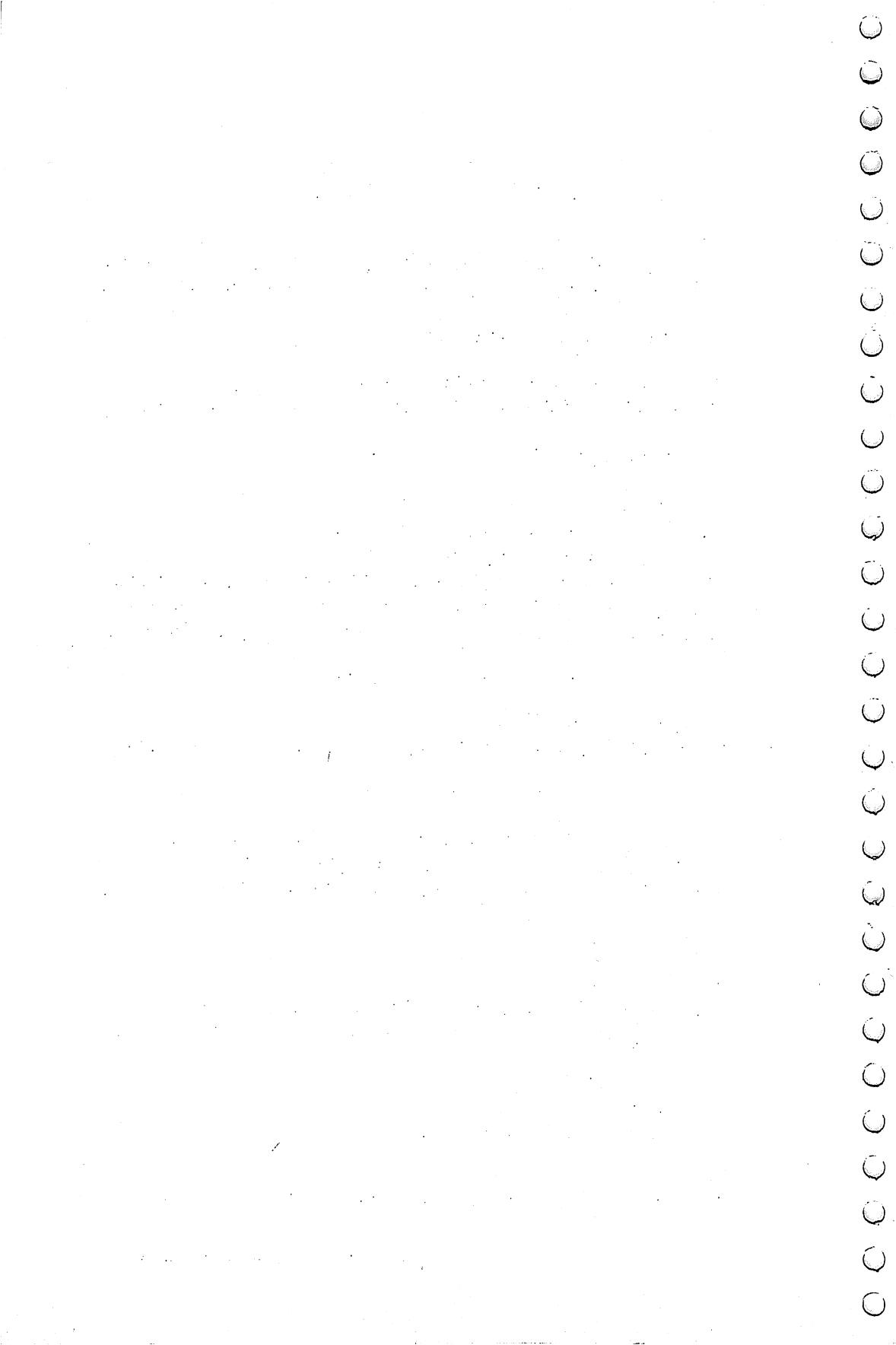
---

17. What keys are used to insert and delete characters in a line?

---

18. When a line is being edited, how is the editing ended?

---



## CHAPTER 3

# COMPUTER ARITHMETIC AND PROGRAM MANAGEMENT

### 3—1 OBJECTIVES

You will need a blank diskette for some of these activities. Instructions for installing the disk drive that you will also need are given in the *VIC-1541 Single Drive Floppy Disk User's Manual*.

#### **Doing Arithmetic on the Computer**

Ultimately, all mathematics on the computer is done using the simplest arithmetic operations. It is essential to have a clear understanding of how these arithmetic operations are done.

#### **Using Parentheses in Computations**

All mathematical expressions must be typed on a single line when you enter them into the computer. Some expressions can be handled this way only by organizing parts of the expressions in parentheses. Thus, the effective use of parentheses is a necessary skill.

#### **Learning E Notation for Numbers**

The computer must deal with both very large and very small numbers. The computer uses E notation to describe such numbers. You need to be able to recognize and interpret E notation since the computer may type out numbers in this form.

### Initializing a Diskette

New or blank diskettes must be prepared or initialized for use on your Commodore 64. You must know how to initialize diskettes correctly in order to store programs.

### Storing and Retrieving Programs

You need to know the commands that permit you to store programs on and retrieve programs from a diskette. You will learn these and other program management commands.

## 3-2 DISCOVERY EXERCISES

If the disk drive has not been installed, you may use the following instructions to install it. You must first turn the computer off. Plug the grey power cable supplied with the disk drive into the three-prong receptacle at the rear of the disk drive cabinet. Plug the disk drive power cable into a wall outlet. Make sure the ON/OFF switches on the disk drive and the computer are OFF. No lights should be on. Plug the black serial cable into the left serial bus socket at the rear of the disk drive cabinet and into the serial port at the right center of the computer. Now turn on the disk drive and then the computer. The computer **must** be turned on last. You can find further installation details and diagrams in the *VIC-1541 Single Drive Floppy Disk User's Manual*. The disk drive will not be used until step 11.

1. Turn on your TV set and computer. Recall from Chapter 2 the following symbols for arithmetic operations:

- + Addition
- Subtraction
- \* Multiplication
- / Division

To review the arithmetic operations, type the following program:

```
100 INPUT A
110 INPUT B
120 LET C=A*B-B/3
130 PRINT C
140 END
```



Display the program and study it briefly. If you were to execute the program and enter 2 for A and 3 for B, what do you think would be displayed?

---

Execute the program, entering 2 and 3 at the prompts, and write down what happened.

---

2. Clear the program in memory. Now type

```
100 LET A=3*3
110 LET B=3↑2
120 PRINT A
130 PRINT B
140 END
```

Display the program and make sure it is correct. Now execute the program and record what was displayed.

---

Compare the numbers printed out to the expressions in the lines where they were computed. See if you can figure out what is taking place.

3. Type

```
100 LET A=3*3*3
110 LET B=3↑3
```

Execute the program. What was displayed?

---

## 4. Type

```
100 LET A=2*2*2*2
110 LET B=2↑4
```

Execute the program. What was displayed?

---

What is the  $\uparrow$  symbol used for?

---

5. Remember from your introductory algebra course (if you haven't had algebra, don't panic!) that if you want to indicate that a number is to be multiplied by itself three times, you can use an exponent. If the number were 2, you would write the expression as

$$2^3$$

How would this expression be written in BASIC using the  $\uparrow$  symbol? (Hint: See steps 2, 3, and 4.)

---

6. Fill in the operators (symbols) that call for the following arithmetic operations:

Division

---

Exponentiation

---

Multiplication

---

7. Clear the program in memory. Type

```
100 LET A=4+2*6
110 LET B=(4+2)*6
120 LET C=4+(2*6)
130 PRINT A
140 PRINT B
150 PRINT C
160 END
```

The two points of this program are (1) the order in which the arithmetic is done, and (2) the effect of the parentheses. If you look closely, you will see that the same numbers and operations are involved in each of the calculations in lines 100, 110, and 120. The only difference in the lines is the grouping with parentheses. Execute the program and record the numbers displayed.

---

Now change lines 100, 110, and 120 as follows.

```
100 LET A=4+2*6/3
110 LET B=(4+2)*6/3
120 LET C=4+(2*6)/3
```

Run the program and record the numbers displayed.

---

Study the program and the numbers the computer displays until you see what is taking place in the program. The computer uses set rules in such situations. You will study these rules later in the chapter.

---

8. Clear the program in memory. Now enter the following program:

```

100 LET A=3*100*100
110 LET B=3*100*100*100
120 LET C=3*100*100*100*100
130 LET D=3*100*100*100*100*100
140 PRINT A
150 PRINT B
160 PRINT C
170 PRINT D
180 END

```

Execute the program and record the output.

---

Can you explain the different form in which the computer displayed the numbers? (Hint: Count the number of zeros in the multipliers in lines 100, 110, and 120).

9. Type

```

100 LET A=4/(100*100)
110 LET B=4/(100*100*100)
120 LET C=4/(100*100*100*100)
125 LET E=4/(100*100*100*100*100)
165 PRINT E

```

Display the program. Execute the program and record the output.

---

Again, can you see what is taking place in the output? Count the zeros in the denominators in lines 100, 110, 120, and 125.

10. If an E shows up in a number displayed by the computer, what does it mean? Explain in your own words.
- 

If you do not fully understand the purpose of the E notation, don't worry. We will return to it later.

11. Now move on to the subject of storing and retrieving programs. Before you can proceed with storing and retrieving programs, you must learn to initialize a diskette. (If you have not installed your VIC-1541 Disk Drive, see the installation instructions at the beginning of this section or in the *VIC-1541 Single Drive Floppy Disk User's Manual*. Be sure to turn the computer off before installing the disk drive.) If you wish to use a tape recorder for program storage, see the *Commodore 64 User's Guide*.
12. Turn off the computer. Turn on the disk drive. Turn on the computer again. Place a diskette in the disk drive with the notch on the left and the label up. Close the door on the disk drive. With the diskette in the drive, type (but do not press `RETURN` )

```
OPEN 15,8,15,"NEW0:BASIC,ZZ"
```

Note: All programs and information stored on this diskette will be lost when the diskette is initialized by pressing `RETURN` .

Now press `RETURN` .

13. The red light on the disk drive will go on and READY will appear on the screen. The drive will whirr for about 70 seconds. When the drive stops whirring and the red light goes out, you are ready to proceed with storing and retrieving programs. Do not proceed to the next step until the red light goes out.
14. Clear the memory and type the following program

```
100 LET A=2
110 LET B=3
120 LET C=A*B
130 PRINT C
140 END
```

15. Display the program to make sure it is correct. Run the program.
16. Now type

```
SAVE"PRODUCT",8 <RETURN>
```

What happened?

---

This transfers the program in memory to diskette under the name PRODUCT. (You may choose any name for a program as long as it has 16 or fewer characters.)

17. Now type

```
LOAD"$",8 <RETURN>
```

to load the directory (\$) of the diskette on the disk drive (8). When READY appears, type

```
LIST <RETURN>
```

to display the directory. You will see

```
0    "BASIC                "ZZ 2A
1    "PRODUCT"            PRG
663 BLOCKS FREE.
READY.
```

This shows that the program PRODUCT is now on the diskette. The PRG stands for program. The number at the left (1) describes the size in blocks (256 characters) of the program PRODUCT. The white line indicates that the diskette has been entitled BASIC, its ID is ZZ, and the operating system version is 2A. (Compare this with the OPEN statement in step 12.)

18. Clear the program in memory by typing

```
NEW <RETURN>
```

Again obtain the directory of programs. (See step 17.) Are the programs on the diskette affected when the program is cleared from memory?

---

19. Type NEW to erase the directory from memory. Type the following program.

```
100 LET D=2*6-8/4
110 PRINT D
120 END
```

Run the program. Think of a name for this program other than PRODUCT. Again, limit the name to 16 or fewer characters. Record the name below.

---

Move the program in memory to diskette under the name you just recorded above. (See step 16.) Now display the program in memory. Is the program you just saved on diskette still in memory?

---

20. Obtain the directory of programs stored on diskette. (See step 17.) Are the two programs you just entered there?
- 

The two programs you just entered should be stored on diskette. The name of the first program is PRODUCT; the name of the second was written in step 19. To simplify the discussion, this second program will be referred to as **PROGRAM 19**. You, of course, must use the name you selected for the second program.

21. To move PRODUCT from diskette to memory, type

```
LOAD"PRODUCT",8 <RETURN>
```

Display this program and verify that it is the right one. Add line 125 to the program in memory as follows

```
125 PRINT "THE PRODUCT IS"
```

Display the program. Run the program. To replace the old program PRODUCT on diskette with the modified PRODUCT in memory, type

```
SAVE"@0:PRODUCT",8 <RETURN>
```

22. Clear the program in memory. Type LIST to see there is no program in memory. Now move **PRODUCT** from diskette to memory. Display the program. Now move **PROGRAM 19** from diskette to memory. Display the program in memory. Which one is there now?
- 

What happened to the program **PRODUCT** that was in memory when you moved **PROGRAM 19** into memory from diskette?

---

23. Move **PRODUCT** into memory with the **LOAD** command. (See step 21.) Remove the program **PRODUCT** from diskette by typing

```
CLOSE 15  
OPEN 15,8,15,"SCRATCH0:PRODUCT"
```

Display the program in memory. When you cleared **PRODUCT** from the diskette, did **PRODUCT** disappear from memory?

---

24. Obtain a directory of the programs stored on diskette. **PRODUCT** should not be there, but **PROGRAM 19** should be. Inspect the listing of programs. Is everything the way it should be? (See step 17.)
- 

25. Note that if you now clear the memory, you will have lost both the copy of **PRODUCT** on diskette (you removed that in step 23) and the copy in memory. Clear the memory. (See step 18.) Display the program in memory. Is anything there?
- 

Try to move **PRODUCT** from diskette to memory. (See step 21.) What happened on the screen and at the disk drive?

---



The blinking red light indicates that a disk drive error has occurred. SAVE and LOAD commands will cause this error light to stop blinking. Clear **PROGRAM 19** from the diskette. (See step 23.) Remove the diskette from the disk drive. Turn off the computer, the TV set, and the disk drive.

### 3-3 DISCUSSION

#### Doing Arithmetic on the Computer

We are concerned with five arithmetic operations. These are addition, subtraction, multiplication, division, and exponentiation (+, -, \*, /, ↑). The first four are certainly familiar to you, and the only difficulty with the last (exponentiation) is the intimidating word used to define the process.

The exponentiation operation is represented by the ↑ symbol. Exponentiation means "raised to the power." Therefore,  $3\uparrow 4$  means "3 raised to the fourth power," which in turn means 3 multiplied by itself four times, giving 81 as the result.

You need a full understanding of the order in which the computer performs arithmetic operations. An example will illustrate the point. Consider the following expression:

$$2+3\uparrow 2/5-1$$

If the computer simply went through the expression from left to right performing operations as it met them, the computer would add 2 and 3 (giving 5), raise 5 to the second power, (giving 25), divide 25 by 5 (giving 5), subtract 1 from that, and arrive at an answer of 4. However, suppose the computer performs addition and subtraction first, then exponentiation, then multiplication and division. This process would give 5 raised to the second power divided by 4. Then the computer would exponentiate giving 25 divided by 4, and arrive at an answer of 6.25.

You could experiment with different rules for the order of arithmetic operations and might get different answers each time. The point is that there are well-defined rules in BASIC for the order and priority of arithmetic operations. Here they are:

The computer performs operations from left to right using the priority rules below.

The priority for arithmetic operations is

1. Exponentiation
2. Multiplication and division
3. Addition and subtraction

Follow the computer as it performs the familiar expression

$$2+3\uparrow 2/5-1$$

The computer scans left to right for exponentiation. Since there is an exponentiation indicated ( $3\uparrow 2$ ), it is done first. Now the expression is

$$2+9/5-1$$

Scanning from left to right, the computer looks for exponentiation, and finding none, scans left to right for the operations with the next highest priority (multiplication and division). The division is therefore done next, with the following result:

$$2+1.8-1$$

Since there are no more multiplications or divisions left in the expression, the computer scans from left to right for addition and subtraction. The addition gives

$$3.8-1$$

and the final subtraction produces the answer of 2.8.

Review the rules for order and priority of arithmetic operations until they become second nature to you. You will look at the rules again when the use of parentheses is discussed in the next section.

### Using Parentheses in Computations

The rules for order and priority of arithmetic operations are not the whole issue, however. Consider the following more complicated example:

$$\begin{array}{c} \text{B} \qquad \text{C} \\ \text{---} \quad \text{---} \\ | \quad | \quad | \quad | \\ \text{---} \text{A} \text{---} \\ | \quad | \\ \text{---} \end{array}$$

$$((2*3+4\uparrow 2)*2+5)*(3\uparrow 2-4)$$

The difference between this expression and the ones you have been studying is the use of parentheses to group parts of the expression. Following this example in detail will show you how the computer attacks the arithmetic.

The computer starts by scanning from left to right and meets the left parenthesis of **B**. It then looks inside to see if there are any other left parentheses and finds one for **A**. The next parenthesis met is a right parenthesis for **A**. At this point, the computer has isolated the first group of operations to be done, the expression in parentheses **A**:

$$2*3+4\uparrow 2$$

The computer evaluates that expression using the order and priority rules. The result is 22 (check it). Now the problem has become

$$\begin{array}{c} \text{B} \qquad \text{C} \\ \text{---} \quad \text{---} \\ | \quad | \quad | \quad | \\ \text{---} \end{array}$$

$$(22*2+5)*(3\uparrow 2-4)$$

On the next scan, the computer isolates parentheses **B**, does the arithmetic inside, and reduces the problem to

$$\begin{array}{c} \text{C} \\ \text{---} \\ | \quad | \\ \text{---} \end{array}$$

$$49*(3\uparrow 2-4)$$

Since only the **C** parentheses are left, the computer does the arithmetic inside, arriving at

$$49*5$$

which after the final multiplication results in the answer of 245.

Thus, if parentheses are nested, the computer works out from the deepest set, working from left to right. When a set of parentheses is removed, the arithmetic operations inside are done according to the order and priority rules already given.

A very good rule of thumb to follow is: If there is any confusion about how the computer will evaluate an expression, use extra parentheses. Too many cannot hurt, but too few certainly can.

### Learning E Notation for Numbers

In BASIC, numbers are sometimes printed out in what is known as E notation. Examples of E notation are 2.145E+06 and 6.032E-07.

It is easy to see why a special notation is needed for either very large or very small numbers. The computer usually prints out nine digits for a number. A problem comes up if you want the computer to print out a variable whose value is, for instance, 45612800000 (11 digits). The computer prints this number as 4.56128E+10, which means that the decimal point belongs 10 places to the right of its present position. A variable whose value is 8954000000000 is printed as 8.954E+12. The E+12 means that the decimal point belongs 12 places to the right. Very small numbers are expressed in the same way. A variable whose value is 0.0000000683 is printed as 6.83E-08. The E-08 means that the decimal point belongs eight places to the left. The table below should help you understand how to convert from decimal to E notation or from E notation back to decimal notation.

Decimal Form	E Notation
2630000	2.63E+06
263000	2.63E+05
26300	2.63E+04
2630	2.63E+03
263	2.63E+02
26.3	2.63E+01
2.63	2.63
0.263	2.63E-01
0.0263	2.63E-02
0.00263	2.63E-03
0.000263	2.63E-04
0.0000263	2.63E-05
0.00000263	2.63E-06

To change from E notation to decimal notation, look at the sign following the E. If the sign is +, move the decimal point to the right as many places as the number after the +. If the sign is -, move the decimal point to the left. To convert from decimal to E notation, just reverse the process.

Don't get tense about E notation; you will rarely use it when setting up programs on the computer. The main reason for bringing up the issue is that the computer may display numbers in E notation, and you should be able to recognize what is happening.

### **Initializing a Diskette**

You format each diskette only one time by placing it in the disk drive with the notch at the left and the label up, closing the disk drive door, typing `OPEN 15,8,15,"NEW0:(diskette name),ZZ"` and pressing `RETURN`. When the disk stops whirring, you are ready to put information onto that diskette. The `OPEN 15,8,15` part of the command establishes a communications path (the 15's) from the computer to the disk drive device (device #8) for the instruction `"NEW0:(diskette name),ZZ"` to pass. The `ZZ` is the diskette ID and may be any two characters.

### **Storing and Retrieving Programs**

When you turn off the computer, you lose the program in memory. If every time you turned on the computer you had to retype your programs, you would get very little work done. Fortunately, the Commodore 64 computer lets you type long or complicated programs, troubleshoot them, and then store them on a formatted diskette for future use. You need only turn on the disk drive, the TV set, and the computer, and then retrieve any programs that you have previously stored.

You might not want to keep a specific program on a diskette forever, and you have a way to clear programs from diskette storage. If you move a copy of a program from a diskette into memory and then clear the copy from the diskette, a copy remains in memory. Also, you can have one program in memory and clear another program from diskette storage without changing the program in memory.

The commands for performing these program management tasks are given below.

- To move a program from memory to diskette, type  
SAVE"(name of program)",8
- To move a program from diskette storage to memory, type  
LOAD"(name of program)",8
- To clear a program from diskette storage, type  
OPEN 15,8,15,"SCRATCH0:(name of program)"
- To display a directory of programs, type  
LOAD"\$",8  
LIST

The simple SAVE command does not allow you to copy over programs already on diskette. To replace a program on diskette with the program in memory, type

```
SAVE"00:(name of program)",8
```

Alternately, you may erase (SCRATCH) the program and then use a simple SAVE command.

Programs are occasionally lost when using diskette storage. You can avoid such losses by keeping backup or spare copies of programs on another diskette.

Occasionally, when using an OPEN statement, a FILE OPEN ERROR will occur. In such a case, type CLOSE 15 and then execute the OPEN statement again.

You can use a tape cassette with a Commodore DATASSETTE tape recorder to save and retrieve programs. The SAVE and LOAD commands for tape cassette storage are discussed in the *Commodore 64 User's Guide*. Tape cassette program storage takes more time and is less reliable than diskette storage.

### 3—4 PRACTICE TEST

1. Write the symbols that are used to signify the following arithmetic operations in BASIC expressions:
  - a. Multiplication

---

b. Exponentiation

---

c. Division

---

2. When evaluating arithmetic expressions without parentheses, the computer follows a priority of operations. What is this priority?

a. 1st:

---

b. 2nd:

---

c. 3rd:

---

3. Which arithmetic operation is performed first when the following direct mode statement is entered?

PRINT (3+5)↑2

---

4. When scanning arithmetic expressions, the computer searches in a specific direction. What is this direction?

---

5. Write a BASIC statement equivalent to the following expression. Number the line 100.

$$A = (4 + 3B/D)^2$$

---

6. If the following program is executed, what will the computer display?

```
100 LET A=2
110 LET B=3
120 LET C=(A*B+2)/2
130 PRINT C
140 END
```

---

7. Convert the following numbers to E notation:

a. 5160000000

---

b. 0.0000314

---

8. Convert the following numbers to decimal notation.

a. 7.258E+06

---

b. 1.437E-03

---

9. Give the order in which the computer will perform the operations in the expression below.

```
100 LET A=(6/3+4)^2
```

---

10. What commands are needed to carry out the following operations:

a. Move a program from diskette storage to memory.

---



b. Move a program from memory to diskette storage.

---

c. Clear a program from diskette storage.

---

d. Clear out a program in memory.

---

e. Display the program in memory.

---

f. Execute the program in memory.

---

g. Display the names of all the files saved on diskette.

---

11. Suppose you are typing a line into the computer and have not yet pressed **RETURN** . How do you correct a single character?

---



## CHAPTER 4

# INPUT, OUTPUT, AND SIMPLE APPLICATIONS

### 4—1 OBJECTIVES

In this chapter you will get down to the business of writing programs. You will also increase your knowledge of BASIC by looking at some details about input and output. The objectives are as follows.

#### **Getting Numbers into a BASIC Program**

There are only three ways to enter numbers into the computer for a BASIC program. Since the computer is concerned mainly with numbers, you need to understand how to input these numbers.

#### **Printing Out Variables and Strings**

After information is computed, it must be printed out. Usually you will want to output strings of characters as well as numbers. String output is handled essentially the same as numbers, but needs special attention.

#### **Spacing the Printout**

The computer spaces output in a standard manner. You will learn how to alter the spacing of output.

#### **Using the REM Statement**

The wise programmer includes comments in programs to help explain or interpret what is being done. The REM (remark) statement in BASIC permits you to do this.

### Working with Program Examples

Your ultimate goal is to learn to write and troubleshoot programs. In this chapter you will begin some modest program assignments.

## 4-2 DISCOVERY EXERCISES

1. Turn on the TV set and the computer. Enter the following program:

```
100 INPUT A
110 INPUT B
120 INPUT C
130 LET D=A+B+C
140 PRINT D
150 END
```

What do you think will happen if you run this program?

---

Run the program. When the first question mark (the INPUT prompt for A) is displayed, type in 2. When the second question mark comes up, type in 3, and finally, at the last question mark, type in 5. Record what happened below.

---

2. Note that in the program in step 1 there are three INPUT statements (lines 100, 110, and 120). Type

```
100
110
```

What does this do to the program?

---

Display the program in memory and see if you are right. Then type

```
120 INPUT A,B,C
```

Display the program. What has happened?

---

3. Run the program, and when the INPUT prompt (?) is displayed, type in

2,3,5

What happened?

---

Can you input more than one variable at a time in a BASIC program?

---

4. Run the program, and when the INPUT prompt is displayed, type

2,3

What happened?

---

What is the computer waiting for?

---

Type

2,3,5

and record below what happened.

---

Notice that the 2 in 2,3,5 is used as the third number while the 3 and 5 are ignored.

5. Run the program and when the INPUT prompt appears, type

2,3,5,1

What happened?

---

6. Can you type more numbers than called for at an INPUT statement?

---

What will happen if you do?

---

7. Can you type in fewer numbers than called for at an INPUT statement?

---

What will happen if you do?

---

8. Type

```
120 READ A,B,C
```

Display the program. What has happened?

---

Run the program and record what happened.

---

9. Now type

```
125 DATA 2,3,5
```

and display the program. What has happened?

---

10. Run the program and record what happened.

---

From the above, you can infer that whenever a BASIC program contains a READ statement, it must also contain another type of statement. What is that statement?

\_\_\_\_\_

11. Name two different methods (other than using a LET statement) for getting numbers into a program. (Hint: See steps 1 and 8.)
- \_\_\_\_\_

12. Display the program in memory. Delete the DATA statement. Type

```
145 DATA 2,3,5
```

and display the program again. What has happened?

\_\_\_\_\_

13. Run the program and record the output.
- \_\_\_\_\_

Does it appear to make any difference where in the program the DATA statement appears?

\_\_\_\_\_

14. Clear the program in memory. Enter the program below

```
100 READ A,B
110 LET C=A/B
120 PRINT C
130 GOTO 100
140 DATA 2,1,6,2,90,9,35,7
150 END
```

What do you think will happen if you run the program?

\_\_\_\_\_

Try it and see if you were correct. Record the output.

---

Is the "OUT OF DATA" message associated with the READ statement or the DATA statement?

---

15. Delete the DATA statement in line 140 from the program. Now enter the following statements:

```
105 DATA 10,2
115 DATA 100,50
125 DATA 50,5
```

Display the program in memory. What has taken place?

---

16. If you run the program, what do you think will be displayed?

---

Run the program and see if you were correct. Record the output below.

---

17. Can you have more than one DATA statement in a BASIC program?

---

Does it seem to make any difference where in the program the DATA statements appear?

---



18. Clear the program in memory. Enter the following program:

```
100 LET A=10
110 PRINT A
120 END
```

What will happen if you run this program?

---

Run the program and record what took place.

---

19. Now type

```
110 PRINT "A"
```

and display the program in memory. What has happened?

---

What will happen if you run the program?

---

Run the program and record what the computer displayed.

---

20. Type

```
110 PRINT "HOUND DOG = ";A
```

and display the program in memory. What do you think will happen if you run this program?

---

Run the program and record what did happen.

---

21. Now let's try a different wrinkle. Type

```
110 PRINT "B = ";A
```

Display the program and study it carefully. If you run the program, what do you think will happen?

---

Try it and see if you were right. Record the output below.

---

22. The next line is longer than the 40 spaces available on one screen line. Do not press **RETURN** when you come to the edge of the screen. Just keep typing. Type

```
90 REM THIS IS A PROGRAM TO DEMONSTRATE READ/DATA
```

Display the program. What has happened?

---

Program lines can be no more than two screen lines (80 characters) long. Run the program. What was the output?

---

Does the REM statement in line 90 have any effect on the program?

---

23. Clear the program in memory. Enter the following program:

```
100 REM METRIC CONVERSION PROGRAM
110 REM CONVERT LBS TO GRAMS
120 PRINT "INPUT NO. OF LBS.";
130 INPUT P
140 LET G=454*P
150 PRINT P;" POUNDS IS ";G;"GRAMS"
160 GOTO 120
170 END
```

Display the program and verify it is correct. Study the program carefully and try to guess what will happen when it is executed. Now run the program. When the INPUT prompt appears, enter any number you desire. Note what is displayed. Repeat this process several times, then jump the computer out of the INPUT loop. If you have forgotten how, see Chapter 2, step 22 of the Discovery Exercises.

What is the purpose of the REM statement?

---

24. Type

```
115 INPUT P
130
160 GOTO 115
```

and then display the program in memory. What has happened?

---

Will the program work in this form?

---

Run the program and at the INPUT prompt, type 1. What happened?

---

Jump the program out of the INPUT loop.

25. Experiment with this program a bit more. Clear the program in memory and enter it again, modified as follows:

```

100 REM METRIC CONVERSION PROGRAM
110 REM CONVERT LBS. TO GRAMS
120 PRINT "INPUT NO. OF LBS.";
130 INPUT P
140 PRINT P;"POUNDS IS ";G;"GRAMS"
150 LET G=454*P
160 GOTO 120
170 END

```

Will the program run correctly in this form?

---

Run the program and at the INPUT prompt, type 2. What happened?

---

Explain in your own words what is wrong. Remember that if a variable is not defined initially in a program, the computer will set it equal to 0.

---

Jump the program out of the INPUT loop.

26. Clear the program in memory. Enter the following program:

```

100 READ A
110 PRINT A
120 GOTO 100
130 DATA 10,12,8,9,112233445566,-82,5,6
140 END

```

Run the program and record what happened.

---

## 27. Type

```
110 PRINT A,
```

Note that all you do is insert a comma after the A in line 110. Execute the program. Are the numbers in the last two columns aligned?

---

## 28. Now replace the comma after A with a semicolon by typing

```
110 PRINT A;
```

Run the program and record what happened.

---

## 29. If a variable in a PRINT statement is not followed by punctuation marks, how does the computer space the output? (Hint: See step 26.)

---

Suppose the variable is followed by a comma?

---

What will happen if the variable is followed by a semicolon?

---

## 30. Clear the program in memory and enter the following program:

```
100 LET A=15
110 READ B
120 PRINT TAB(A);B;
130 LET A=A+10
140 GOTO 110
150 DATA 1,2,3
160 END
```

Run the program and record what happened.

---

## 31. Type

```
130 LET A=A+5
```

Run the program and record what happened.

---

## 32. Type

```
130 LET A=A+20
```

Run the program and record what happened.

---

## 33. What does the TAB in the print statement appear to control?

---

## 34. This concludes the computer work for now. Turn off the computer and the TV set.

### 4-3 DISCUSSION

In this chapter you have begun to get away from the mere mechanics of controlling the computer. Instead, you have begun to concentrate on writing and troubleshooting programs. This skill doesn't come naturally to most people, and consequently we will give the topic a great deal of attention, both now and in later chapters.

#### Getting Numbers into a BASIC Program

In Chapter 2 you learned one way to get numbers into a program: assigning values to a variable in the program itself. For example,

```
100 LET A=6
```

introduces the value 6 into a program and stores the number under the variable name A. This method has limitations, so you need to know other ways to introduce numbers into a BASIC program. Let's

look first at the INPUT statement and how it is used. An example might be

```
260 INPUT G
```

When the computer executes this line, it displays a question mark as a prompt that input is expected, then it waits for you to type in a number. In the case above, the number typed in is known as G.

A single INPUT statement can call for more than one variable. For instance,

```
420 INPUT A,B,C,D
```

In this case, the computer displays the same INPUT prompt (the question mark) but now the computer expects you to type in four numbers separated by commas. If you enter only three numbers (or fewer) and press the RETURN key, the computer displays ??. If you type in more than four numbers, **EXTRA IGNORED** will appear on the screen.

One last method of providing numerical input to the computer is with the READ and DATA statements. The computer handles the READ statement

```
100 READ A,B,C,D
```

in the same way as the INPUT statement, with two exceptions. First, the computer does not stop. There is no need to, as you will see. Second, the computer reads the numbers to be input from DATA statements contained within the program rather than displaying the INPUT prompt and waiting for you to enter those numbers.

The following program illustrates the READ and DATA statements.

```
100 READ A,B,C,D
110 LET E=A+B+C+D
120 PRINT E
130 DATA 25,3,17,12
140 END
```

The program reads four numbers from the DATA statement and prints out the sum of the numbers. It makes no difference where in the program the DATA statement appears. Also, there can be

more than one DATA statement, and DATA statements need not be grouped together at the same place in the program. When the READ statement calls for numbers, the computer takes them in order from the DATA statements, beginning with the lowest numbered statement. If a READ statement still requests numbers after the computer exhausts all the numbers in the available DATA statements, the computer types an "out of data" message and stops.

In summary, there are three methods for introducing numbers into BASIC programs. They are: (1) LET statements, (2) INPUT statements, and (3) READ and DATA statements. Each of these methods can be used to advantage. You will become familiar with the advantages and disadvantages of each method as you spend more time writing programs.

- **You can put numbers in a BASIC program with:**  
(1) LET (assignment), (2) INPUT, and (3) READ/DATA statements.

### **Printing out Variables and Strings**

Output from the computer is quite simple. The computer can print out either the numerical value of a variable (a number) or a string of characters. For instance, suppose there is a variable named X, and the number 2 is stored in the X location in memory. The program

```
100 LET X=2
110 PRINT "X"
120 PRINT X
130 END
```

shows the difference between string and variable output. Line 110 prints the character X, since X is enclosed in quotation marks. Line 120 prints 2, since that is the number stored in location X.

The rule is clear. Characters within quotation marks are strings, and strings are printed out exactly as input (but without the quotation marks). The computer does not attempt to analyze or detect what is within the quotation marks, it simply prints the characters. If a variable in a PRINT statement is not contained within quotations, the computer evaluates the expression and prints out the numerical value of that variable.

It is possible to do computation within a PRINT statement. Thus the statement



```
100 PRINT A+B+C,D
```

will cause the computer to print the sum of the numbers stored in **A**, **B**, and **C**, then the number stored in **D**. Of course, you must assign values to the variables **A**, **B**, **C**, and **D** beforehand or the computer will assign zeros for their values.

### Spacing the Printout

Commodore 64 BASIC has a built-in standard spacing mechanism that prints numbers at fixed places on the line. When quantities in a **PRINT** statement are separated by commas, the computer uses standard spacing. The four standard printing positions are 1, 11, 21, and 31. The first place in each number is set aside for a minus sign whether it is needed or not. The comma signals the computer to move to the next print position on the line. If the computer is already at the last position on a line and encounters a comma in a **PRINT** statement, it does a carriage return and prints the number on the first position on the next line. Thus, if **A**, **B**, **C**, **D**, and **E** are small integers

```
100 PRINT A,B,C,D,E
```

would cause the numerical values of **A**, **B**, **C**, and **D** to be printed evenly spaced across a line in the four standard positions. The numerical value of **E** would be printed below the value of **A** on the next line. If **A**, **B**, **C**, **D**, and **E** are numbers that require a longer **E** notation, then **A** and **B** will be printed on the first line; **C** and **D** on the second in two standard positions while **E** will be printed on the third line under **A** and **C**.

### ■ Commas in **PRINT** statements produce aligned columns.

When used in a **PRINT** statement, a semicolon between variables instructs the computer to space in a nonstandard manner. For instance, the output caused by the statement

```
100 PRINT A;B;C;D;E
```

will be spaced closer together than the output of a **PRINT** statement with commas. The semicolon instructs the computer to place negative numbers one space apart and positive numbers two spaces apart.

Two or more numbers may be placed on a line depending on the size and format of the numbers. It is enough to realize that the statement

```
100 PRINT A;B
```

produces closer spacing of output than the statement

```
100 PRINT A,B
```

Finally, you can control the spacing of a line more precisely by using the TAB function in PRINT statements. The TAB function works the same way as a tabulator setting on a typewriter. There are 40 tab positions (0-39) available on a screen line starting at tab position 0 or printing position 1. The tab position is one less than the printing position.

The statement

```
100 PRINT TAB(5);A;TAB(25);B
```

signals the computer to space to the 5th tab position or 6th printing position, print the numerical value of **A**, space to the 25th tab position or 26th printing position, and print the numerical value of **B**. If **B** has too many digits to fit on the line, the computer will place **B** at the beginning of the next line if a comma comes before **B**. The extra digits of **B** will be placed on the next line if a semicolon comes before **B**.

It is also possible to use a variable tab setting that is controlled by the computer:

```
100 PRINT TAB(X);A
```

This statement causes the computer to look up the value of **X**, then space to the printing position determined by the largest integer in **X** (for example, if  $X = 23.1435$ , the computer will space to the 23rd printing position), then print the numerical value of in the 24th printing position.

■ **Use the TAB function to produce variable spacing.**

You can use an empty PRINT statement, for instance,

**100 PRINT**

to produce vertical spacing in the output. The computer looks for the quantity to be printed and finds none. It then looks for punctuation, and finding none, orders a carriage return and advances the cursor one line. If you want two or three empty lines in a printout, use two or three empty PRINT statements.

**Using the REM Statement**

The REM (for "remark") statement is quite different from the statements you have seen previously. As soon as the computer senses the characters REM following the line number, it ignores the balance of the statement and goes on to the next line. The REM statement provides information for the benefit of the programmer or someone reading the program. REM statements make it much easier to follow what is taking place in the program. The wise programmer uses REM statements liberally.

To see the value of REM statements, read the following two programs. The programs produce identical results, but the second program uses REM statements to describe what is happening. You can be the judge of which program is easier to follow.

1. Without REM statements:

```
100 INPUT A,B,C,D
110 LET X=(A+B+C+D)/4
120 PRINT X
130 END
```

2. With REM statements:

```
100 REM COMPUTE THE AVERAGE OF FOUR NUMBERS
110 REM INPUT THE FOUR NUMBERS
120 INPUT A,B,C,D
130 REM COMPUTE THE AVERAGE
140 LET X=(A+B+C+D)/4
150 REM PRINT OUT THE AVERAGE
160 PRINT X
170 END
```

- Put REM statements in programs to make them easy to read.

#### 4-4 WORKING WITH PROGRAM EXAMPLES

In later chapters, you will spend progressively more time writing and debugging programs. The examples in this chapter are very simple but illustrate the ideas you have studied. Study each example carefully until you are certain you understand all the details. You might want to enter the programs into the computer and execute them to verify that they work as intended.

##### Example 1 - Unit Prices

Your problem is to write a program to compute unit prices on supermarket items. Let **T** stand for the total price, **N** for the number of items, and **U** for the unit price. You can compute the unit price as follows:

$$U = T/N$$

As an example, suppose that a case of 12 large cans of fruit juice costs \$6.96. The unit cost per can would be

$$U = 6.96/12 = \$0.58$$

The program can be designed to execute the following output:

```
WHAT IS THE TOTAL PRICE? (You enter value of T)
WHAT IS THE NUMBER OF ITEMS? (You enter value of N)
UNIT PRICE IS (Computer displays value of U)
```

You can examine this example to see how the program relates to the desired output.

```
WHAT IS THE TOTAL PRICE (Enter T)
      100                                200
WHAT IS THE NUMBER OF ITEMS (Enter N)
      300                                400
500: (Compute unit price)
UNIT PRICE IS (Output U)
      600
700: (End program)
```

You can write each line of the program so that the numbers below the statements will be the line numbers in the program. In line 100 you want the computer to type out the message indicated. Use the PRINT command for this purpose.

```
100 PRINT "WHAT IS THE TOTAL PRICE ";
```

Note the semicolon outside the closing quotation marks. It is there so that there will be no carriage return. Instead the INPUT prompt will appear at the end of the printed line. Line 200 should be an INPUT statement to call for the input of T.

```
200 INPUT T
```

The message in line 300 requires a PRINT statement.

```
300 PRINT "WHAT IS THE NUMBER OF ITEMS ";
```

The input for the total number of items is handled the same as the total price.

```
400 INPUT N
```

The unit price is computed in line 500.

```
500 LET U=T/N
```

The next line is a message, followed by the unit price, and is handled with a PRINT statement.

```
600 PRINT "UNIT PRICE IS ";U
```

Finally, there is an END statement.

```
700 END
```

Now pull the whole program together.

```

100 PRINT "WHAT IS THE TOTAL PRICE ";
200 INPUT T
300 PRINT "WHAT IS THE NUMBER OF ITEMS ";
400 INPUT N
500 LET U=T/N
600 PRINT "UNIT PRICE IS ";U
700 END

```

Study the program to make sure you see the purpose of each line as related to the original statement of the problem.

### Example 2 - Converting Temperature

The relationship between temperature measured in degrees Fahrenheit and in degrees Celsius is

$$C = (5/9)(F - 32)$$

where **C** stands for degrees Celsius and **F** stands for degrees Fahrenheit. If, for example, **F** is 212, then **C** is

$$C = (5/9)(212 - 32) = 100$$

As in the previous example, the first thing to consider is how you want the output to appear. Suppose you want to see the following:

```

INPUT NO. OF DEGREES F
? (You enter value of F)
(Value of F) DEGREES F IS (Answer) DEGREES C

```

Again, you can consider the output in parts that will be generated by the lines in the program:

```

INPUT NO. OF DEGREES F
100
200: (Enter F)
300: (Compute degrees C)
(Output F) DEGREES F IS (Output C) DEGREES C
400
500: (End program)

```

The corresponding program is

```

100 PRINT "INPUT NO. OF DEGREES F "
200 INPUT F
300 LET C=(5/9)*(F - 32)
400 PRINT F;" DEGREES F IS ";C;" DEGREES C"
500 END

```

This program is slightly different from the one in the first example. In line 100, there is no punctuation following the string. Thus the INPUT prompt generated by the INPUT statement in line 200 will be printed on the line following the initial string. The PRINT statement in line 400 prints out (1) the value of F, (2) a string, (3) the value of C, and (4) a second string. The semicolons in line 400 set off the variables and strings in the PRINT statements.

### Example 3 - Sum and Product of Numbers

The final example in this chapter is a program that computes the sum and product of any two numbers. Again, consider how you want the output to appear.

```

INPUT A? (You enter value of A)
INPUT B? (You enter value of B)
SUM OF A AND B IS (Computer prints sum)
PRODUCT OF A AND B IS (Computer prints product)
(Computer inserts blank lines.)
INPUT A? (You enter value of A)
(etc.)

```

Let's proceed more rapidly with this problem. The first line of the desired output can be handled by the following two statements:

```

100 PRINT "INPUT A ";
110 INPUT A

```

Note carefully that the message printed in line 100 is window dressing for the program and has nothing to do with the calculations. The input instruction that is important to the computer is in line 110. You can generate the second line of the desired output as follows:

```

120 PRINT "INPUT B ";
130 INPUT B

```

The sum and product of the two numbers are printed out as follows:

```
140 PRINT "SUM OF A AND B IS ";A+B
150 PRINT "PRODUCT OF A AND B IS ";A*B
```

So far, the program computes the sum and product of two numbers and stops. You can make the program continue computing by adding a loop (a GOTO). Also, you can separate the outputs by including blank lines generated by empty PRINT statements, as follows:

```
160 PRINT
170 PRINT
180 GOTO 100
```

Of course, the final line should be the END statement.

```
190 END
```

The whole program is listed below.

```
100 PRINT "INPUT A ";
110 INPUT A
120 PRINT "INPUT B ";
130 INPUT B
140 PRINT "SUM OF A AND B IS ";A+B
150 PRINT "PRODUCT OF A AND B IS ";A*B
160 PRINT
170 PRINT
180 GOTO 100
190 END
```

As structured, the program will keep looping back until you jump the program out of the INPUT loop.

You could have computed the sum and product of the two numbers in separate lines, as in the following version of the same program.

```
100 PRINT "INPUT A ";
110 INPUT A
120 PRINT "INPUT B ";
130 INPUT B
140 LET S=A+B
150 PRINT "SUM OF A AND B IS ";S
160 LET P=A*B
```



```
170 PRINT "PRODUCT OF A AND B IS ";P
180 PRINT
190 PRINT
200 GOTO 100
210 END
```

A final note on programming: Work out the details of a program with paper and pencil before you type the program into the Commodore 64 computer. You will save yourself time and trouble.

#### 4-5 PROBLEMS

1. Write a program that will read the four numbers 10, 9, 1, and 2 from a DATA statement, putting the numbers in A, B, C, and D, respectively. The program should add the first two numbers and put the sum in S, then compute the product of the last two numbers and put the result in P. Finally, the program should print the value of S and P on the same line.
2. Write a program that will call for the input of four numbers, then print back the numbers in reverse order. For example, if you type in 5, 2, 11, 12, the computer should type back 12, 11, 2, 5. The program must work for any set of four numbers that you decide to type in. Use only four lines including the END statement in your program.
3. Write a program to read variables A, B, C, and D from numbers of your choice in a DATA statement. Have the numbers print out vertically with B below A, C below B, and D below C.
4. What will be output if you run the following program?

```
100 READ X,Y,Z
110 DATA 2,5,3
120 LET T=X*Y+Z
130 LET S=Y↑2
140 PRINT T,S
150 END
```

5. What is wrong with this program?

```

100 LET A=2
110 READ B
120 LET A=A+C/B
130 DATA 3
140 PRINT A
150 END

```

6. Explain in your own words what the following program does.

```

100 INPUT A,B
110 LET S=A+B
120 LET T=A-B
130 LET U=A*B
140 PRINT S;T;U
150 END

```

7. One of the ratios used to judge the health of a business is the acid-test ratio. To figure the acid-test ratio, divide the sum of cash, marketable securities, and receivables by current liabilities. Write a program to call for the input of the necessary quantities, then compute and output the acid-test ratio.
8. Write a program to count and print out by fives, beginning with zero. The first few numbers will be 0, 5, 10, 15, and so on. Interrupt the program manually when 40 or 50 numbers have been printed out.
9. The intended output of the program below is 1, 3, 5, 7, 9, and so forth. The program below has an error. What is wrong?

```

100 LET A=1
110 PRINT A;
120 LET A=A+2
130 GOTO 100
140 END

```

10. If an object is dropped near the surface of the earth, the distance it will fall in a given time is

$$S = 16T^2$$

where **S** is the distance fallen (in feet) and **T** is the time of fall in seconds. Write a program that, when executed, will produce the following output:

```
TIME OF FALL (SEC) ? (You enter T)
OBJECT FALLS (Computer types out S) FEET
```

11. The volume of a box is computed as

$$V = LWH$$

where **L**, **W**, **H** are the length, width, and height, respectively. If these are all measured in centimeters, for example, the volume will be in cubic centimeters. The object is to write a program that will produce the following output when executed:

```
LENGTH (CM) ? (You enter L)
WIDTH (CM) ? (You enter W)
HEIGHT (CM) ? (You enter H)
VOLUME IS (Computer types out V) CUBIC CM.
```

The program below is incorrect and will not produce the output called for above. What is wrong?

```
100 PRINT "LENGTH (CM)";L
110 PRINT "WIDTH (CM)";W
120 PRINT "HEIGHT (CM)";H
130 INPUT L,W,H
140 LET V=L*W*H
150 PRINT "VOLUME IS";V;"CUBIC CM."
160 END
```

12. The program below calls for two numbers, **A** and **B**, in the INPUT statement. The problem is to supply the missing statements so that when **A** and **B** are printed out, their values are interchanged.

```
100 INPUT A,B
110
120
130
140 PRINT A,B
150 END
```

13. Suppose the odometer on your car reads  $R_1$  miles when the gas tank is full. You drive until the odometer reading is  $R_2$ , at which point  $G$  gallons of gasoline are required to fill the tank. Miles per gallon are expressed by

$$M = (R_2 - R_1)/G$$

Write a program that uses the following data to compute the mileage.

$R_1$	$R_2$	$G$
21423	21493	5
05270	05504	13
65214	65559	11.5

14. Suppose you are writing a program to print out numbers on a form. Assume there are three numbers to be handled, and they are in a DATA statement. Write a program that will print the first number beginning 20 positions in from the left of the output device. Follow this with two blank lines. Print the second number beginning at position 10. Then print a blank line, followed by the third number beginning at position 15. Put any numbers you wish in the DATA statement and try out the program.
15. It is known that a DATA statement contains examination grades for a class of 10 students. Write a program containing no more than four statements, counting the DATA and END statements, to compute and print out the class average. Try out the program on sample data of your choice.
16. An old tale tells of a wise man who as a reward for inventing the game of chess asked to receive 1 grain of wheat on the first square of the chess board, 2 grains on the second, 4 grains on the third, 8 on the fourth, and so on. Write a program to print out the number of the square and the number of grains on that square. The program should have a GOTO statement that creates a loop. How many grains of wheat will be on the sixty-fourth square? Run the program and find out.

17. If compound interest is paid, the true annual interest rate is higher than the nominal rate quoted for the investment. The following BASIC formula computes this true annual interest rate:

$$T = ((1 + R/(100 * M))^M - 1) * 100$$

In this expression, T is the true annual interest rate in percent, R is the nominal interest rate in percent, and M is the number of times the investment is compounded per year. Write a program that will produce the following output when executed:

```

QUOTED INTEREST RATE (PERCENT)
? (You enter rate)
NUMBER OF TIMES COMPOUNDED PER YEAR
? (You enter times)
TRUE ANNUAL INTEREST RATE IS
(Computer types out answer)

```

18. Simple interest on an investment is computed according to the following rule:

$$I = (P)(R/100)(T/365)$$

where P is the principal invested at an annual interest rate R (expressed in percent) for a time T (expressed in days). Write a program that will generate the display shown below

```

WHAT IS THE PRINCIPAL
? (You type in the principal)
WHAT IS THE ANNUAL INTEREST RATE (%)
? (You type in the interest rate)
WHAT IS THE TERM IN DAYS
? (You type in the term)

```

```

FOR AN INVESTMENT OF
(Computer types out the principal)
AT AN ANNUAL INTEREST RATE OF
(Computer types out the rate)
PERCENT INVESTED FOR
(Computer types out the term)
DAYS, THE INTEREST IS
(Computer types out the interest)

```

19. If an amount of money  $P$  is left to accumulate interest at a rate of  $I$  percent per year for  $N$  years, the money will grow to a total amount  $T$  given by

$$T = P * (1 + I/100)^{\uparrow N}$$

As an example, if  $P = \$1000$ ,  $I = 6$  percent, and  $N = 5$  years,

$$T = 1000 * (1 + 6/100)^{\uparrow 5} = 1338.23$$

Write a program that when executed will produce the following output:

```
INITIAL INVESTMENT ? (You enter P)
ANNUAL INTEREST RATE (%) ? (You enter I)
YEARS LEFT TO ACCRUE INTEREST ? (You enter N)
TOTAL VALUE IS (Computer types out T)
```

20. If an amount of money  $P$  is left to accumulate interest at  $I$  percent compounded  $J$  times per year for  $N$  years, the value of the investment will be

$$T = P * (1 + I/(100 * J))^{\uparrow (J * N)}$$

Write a program that will call for the input of  $P$ ,  $I$ ,  $J$ , and  $N$ . Execute the program as needed to get the value of \$1000 invested at 8 percent for 2 years compounded

- a. Annually ( $J = 1$ )
- b. Semiannually ( $J = 2$ )
- c. Monthly ( $J = 12$ )
- d. Weekly ( $J = 52$ )
- e. Daily ( $J = 365$ )

A savings and loan company launches a big campaign advertising the fact that it compounds interest every day instead of each week. How great an incentive is there for transferring your money to that savings and loan?

**4-6 PRACTICE TEST**

1. What will be output if you run the following program?

```
100 LET X=1
110 PRINT X;
120 LET X=X+1
130 GOTO 110
140 END
```

---

2. Describe three ways to introduce numbers into a BASIC program.

---

3. In a PRINT statement, what is a collection of characters between quotation marks called?

---

4. What is the purpose of the REM statement?

---

5. If there is a READ statement in a BASIC program, what other type of statement must also be present in the program?

---

6. What will happen if the following program is executed?

```
100 LET X=3
110 LET Y=4
120 PRINT "Y = ";X
130 END
```

---

7. How many standard print columns per line are provided for in BASIC when the print quantities are separated by commas?

---

8. How many DATA statements can there be in a program?

---

9. What is the purpose of the TAB function in BASIC?

---

10. What will the output look like if the following program is executed?

```
100 LET A=1
110 LET B=3
120 PRINT A,B
130 PRINT A;B
140 END
```

---

11. The program

```
100 INPUT A,B
110 LET C=A+B
120 PRINT C
130 END
```

is executed, and in response to the INPUT prompt you type the numbers 10, 12, and 13. Describe exactly what will happen.

---



12. If you multiply miles by 1.609, you convert them to kilometers. Thus, 10 miles equal 16.09 kilometers, and so on. Write a program that will produce the following output.

```
INPUT NO OF MILES? (You type in a number)
(Computer types your number) MILES EQUAL (Answer)
KILOMETERS
```

---



# **DECISIONS, BRANCHING AND APPLICATIONS**

## **5—1 OBJECTIVES**

The power of the computer rests in large part on its ability to make decisions about quantities in programs. In this chapter you will explore this capability and will continue learning to write BASIC programs. The objectives are as follows.

### **Understanding Transfer Statements**

The information in a program line can make the computer decide to jump to line numbers out of numerical order. Such a transfer to a program line may be unconditional or may depend on values of variables in the program. These conditional and unconditional transfer statements can make simple programs produce powerful and useful results.

### **Working with Program Examples**

As in the previous chapter, you will write BASIC programs that apply what you learned.

### **Finding Errors in Programs**

When first written, almost all programs have errors. Troubleshooting programs is a vital skill that, like programming itself, can be learned.

**5-2 DISCOVERY EXERCISES**

1. Turn on the TV set and the computer. Enter the following program:

```
100 LET X=1
110 PRINT X
120 LET X=X+1
130 IF X<5 THEN 110
140 END
```

The < symbol in line 130 means "less than"; thus, the statement means "If X is less than 5, then 110." Study the program carefully. What do you think the computer will print if you run the program?

---

Run the program and record the output.

---

2. Now type

```
100 LET X=2
```

Display the program in memory. What will the output be now?

---

Run the program and write down what the computer printed.

---

3. Now make another change in the program to see if you are following what is taking place. Type

```
120 LET X=X+2
```

Display the program and study it carefully. What do you think the computer will do now?

---

Run the program and see if you were right. In the space below copy what appeared on the screen.

---

4. To explore another idea related to the program in memory, you need to make some changes. Edit the program to make it just like the one below or clear the program in memory and enter the one below.

```
100 LET X=1
110 PRINT X
120 LET X=X+1
130 IF X>=5 THEN 140
135 GOTO 110
140 END
```

Run the program and record the output.

---

Compare the output you recorded above to the output you recorded in step 1. Is there any connection?

---

5. Display the program in memory. Look at the assertion in line 130,  $X \geq 5$ , which means "X is greater than or equal to 5." If X had the value 6, the assertion would be true. If X had the value 3, the assertion would be false.

Now look at the program in step 4 above. When executing the program, the computer starts with line 100, then goes to lines 110, 120, and 130. If the assertion in line 130 is true, which line number will the computer execute next?

---

6. So far, you have seen only two conditions in programs. They are

< (Less than)  
>= (Greater than or equal to)

How would you write the conditions for

Greater than?

---

Less than or equal to?

---

Equal to?

---

Can you guess how to write "not equal to"?

---

If you don't know how to fill in the blanks above, don't worry. We will review everything later. The important thing now is how the IF THEN statement works.

7. Now on to some applications using IF THEN statements. Clear the program in memory and enter the following program:

```
100 PRINT "INPUT EITHER 1, 2, OR 3 ";
110 INPUT Y
120 IF Y=1 THEN 150
130 IF Y=2 THEN 170
140 IF Y=3 THEN 190
150 PRINT "BLOOD"
160 GOTO 100
170 PRINT "SWEAT"
180 GOTO 100
190 PRINT "TEARS"
200 GOTO 100
210 END
```

Display the program and check that you have entered it correctly. Study the program briefly. Remember that when you run the program and the INPUT prompt appears on the screen, you are supposed to type either 1, 2, or 3. Which value of Y will let the computer reach line 150 in the program?

---

Which value or values of Y will let the computer reach line 170?

---

How about line 190?

---

8. Suppose you wanted the computer to type SWEAT. What value of Y should you enter?
- 

See if you were right. Run the program and enter the number you wrote above. What happened?

---

9. What value of Y will cause the computer to type BLOOD?
- 

How would you make the computer type TEARS?

---

Check your responses to see if you were right.

10. The program assumes that the user will type either 1, 2, or 3 when the INPUT prompt appears. Think about the program a bit, then try to figure out what will happen if you type 4 in response to the input prompt. What do you think will happen?
-

Run the program, type 4 in response to the input prompt, and record below what happened.

---

You can easily explain what happened. Press `RUN/STOP|RESTORE` and type LIST to display the program. Consider what the computer does when it encounters an assertion in the IF THEN statement. Remember, if the assertion is true, the computer goes to the line number following the THEN. If the condition is false, the computer goes to the next higher line number.

11. `RUN/STOP` or `RUN/STOP|RESTORE` interrupts a program, but you can also write a program so that the user can end it with a single keystroke. Clear the program in memory and type in the following program:

```
100 PRINT "PRESS SEVERAL KEYS"
110 GET A$
120 IF A$= "Q" THEN 150
130 PRINT A$;
140 GOTO 110
150 PRINT "DONE"
160 END
```

Display the program. Run the program. Press and release several keys before you press Q. What happened?

---

How many times is each character displayed?

---

Press Q to stop the program. Is Q printed on the screen?

---

12. Delete line 130 and delete line 150 of the program in memory. Add line 125 as follows:

```
125 IF A$="" THEN 110
```

Display the program.



13. Now you will use another form of the IF THEN statement. Change line 120 as follows.

```
120 IF A$<> "" THEN PRINT A$
```

Display the program. Run the program. Press several keys. What happens after you press a key?

---

Does the computer indicate that it is awaiting a keystroke?

---

Press the spacebar several times. Then press K. Does the spacebar create spaces?

---

Interrupt the program with RUN/STOP.

14. Delete line 160, the END statement. Now add line 115 as follows:

```
115 IF A$="Q" THEN END
```

Think about the program before you run it. What key would you press to end the program? Run the program and press that key. Were you right?

---

Use the editing features to replace the Q in line 115 with a space, as follows:

```
115 IF A$=" " THEN END
```

What would you press to end the program now?

---

Display the program. Now run the program and press several keys before you press the spacebar. Were you right?

---

15. Clear the memory and type in the following program:

```
100 LET S=0
110 INPUT Y
120 IF Y=11111 THEN 150
130 LET S=S+Y
140 GOTO 100
150 PRINT S
160 END
```

This program is supposed to add up numbers you type in. When you type 11111, the computer displays the sum. The number 11111 is not part of the sum.

16. Run the program, and each time the INPUT prompt appears, type in one number from the following sequence of numbers (remember to press RETURN each time you input one of the numbers).

3      1      6      5      11111

What value is printed out for S?

---

Is the value of S the sum of the numbers you typed in?

---

17. Try to find out why. Type in the following line to study the changing values of S.

```
135 PRINT " S= ";S
```

Display the program. Run the program and input the same values as in step 16. The problem with the program still exists. Compare the values of S printed by line 135 to the values input in line 110 to see why.

---

Though you may have already discovered the logical error in the program, trace the execution of the program by going through the program as if you were the computer. Do this for the first one or two input values and then for the last two input values. Observe that the GOTO statement in line 140 goes to the wrong line, that is, line 100 (where *S* is reset to zero each time).

18. Delete line 135 and EDIT line 140 as follows

```
140 GOTO 110
```

Run the program and type in the same values (3, 1, 6, 5, 11111). Is the output the sum of the numbers you input?

---

Run the program a final time with the same values to verify its correctness.

19. Turn off the computer and the TV set. Go on to the discussion of the objectives.

### 5—3 DISCUSSION

#### Understanding Transfer Statements

From the very beginning of the book, you have been using unconditional transfer statements. The following program illustrates the use of the unconditional transfer statement:

```
100 LET Z=2
110 PRINT Z
120 LET Z=2*Z
130 GOTO 110
140 END
```

Recall that when the computer executes a BASIC program, it goes to the statement with the lowest line number and then executes the statements in increasing line number order. The only way to alter this order is with a transfer statement (or, as you will see in the next chapter, a loop command). The computer executes the program above in this line number order: 100, 110, 120, 130, 110, 120, 130, 110, 120, 130, and so on. The point is that the statement in line

130 causes the computer to jump back to line 110 instead of going to 140.

Note that there are no conditions attached to the statement in line 130. For this reason the GOTO statement is called an unconditional transfer statement. It is also clear that in this case the GOTO statement creates a loop from which there is no escape. The only way to get the computer out of the loop is to interrupt the program from the keyboard while it is running. (The program will eventually produce a number larger than BASIC can handle, and an overflow error will stop the program.)

To sum up, if at some point in a program you want the computer to make an unconditional jump to another line, use the GOTO statement. However, be careful that you don't get the program "hung up" in a loop.

#### ■ GOTO is unconditional.

Some transfer statements contain conditions. By now you have most likely established the connection between the IF THEN statements you saw in the computer work and the notion of the conditional transfer statement.

#### ■ IF THEN statements are conditional.

All conditional transfer statements take the same form.

Line# IF <relation> <condition> <relation> THEN Line#

Notice how the sample conditional statement below follows the form:

```
240 IF 3*X - 2 > Y - Z THEN 360
```

No matter what the assertion, all IF THEN statements have the same format. The IF and THEN as well as the two line numbers in the statement require no special explanation. The heart of the statement is its assertion: the two expressions separated by the condition. Look at the assertion very carefully.

Except for the assertion above, the assertions you have seen have been simple expressions using variables or constants. This type of assertion appears most often in programs. Examples are

```
100 IF U<3 THEN 250
```

```
340 IF S>T THEN 220
```

Sometimes, however, you might want to use more complicated expressions in the IF THEN statements. In the example

```
240 IF 3*X - 2 > Y - Z THEN 360
```

the first relation is

$$3*X - 2$$

which is fine providing that X has a value. The second relation,

$$Y - Z$$

is also fine if Y and Z have values.

To picture what takes place in a program, suppose that X has the value 1, Y is 10, and Z is 4. The computer will simplify the statement by substituting the values of X, Y, and Z and then doing the calculations. The statement then becomes

```
240 IF 1>6 THEN 360
```

Sooner or later, all IF THEN statements are reduced to this form, and the computer judges whether an assertion established by two numbers and a condition is true or false. In this case, the assertion  $1 > 6$  is false because 1 is not greater than 6. However, the computer would judge the assertion  $4 < 10$  as true. If the assertion is true, the program **branches**: It sends the computer to the line number following THEN. If the assertion is false, the computer goes to the next higher line number in the program.

- IF true, THEN branch.
- IF false, THEN go to next higher line number.

Several conditions may be used in the IF THEN statement. The conditions and their meaning are listed below.

Condition	Meaning
=	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to

The "statement" after the THEN or in conditional statements need not be a line number. For instance, these statements are possible:

```
IF X=5 THEN END
IF X>=6 THEN PRINT "YOU LOSE"
```

Most often, however, line numbers follow THEN. The reason is that the ability to branch anywhere in a program is a powerful tool for the programmer.

#### 5-4 WORKING WITH PROGRAM EXAMPLES

Up to this point the programs you have seen have had serious limitations. If the program involved repetition you had no way to stop the process except to interrupt the program. Programs that had no repeating elements were often trivial. The goal is to have the program accomplish a useful task (which may involve repetition) and then shut itself off. The conditional transfer statements give you a way to do just that. Now explore several programs; the first is very simple, but the others give you a taste of how you can make programs work for you.

##### Example 1 - Printout of Number Patterns

Your problem is to write a program that will print out the following number pattern when executed:

```
 2   3   4   5
 6   7   8   9
10  11
```

You need to think about several characteristics of this pattern when you write the program. The first number is 2, and succeeding numbers are spaced across in the standard spacing (four numbers to a line). Each number is 1 greater than the previous one. The last number printed out is 11, then the computer should stop.

Several solutions are possible. The following program is not very elegant but would still work:

```
100 PRINT 2,3,4,5,6,7,8,9,10,11
110 END
```

You might check this program to see that it does in fact produce the correct number pattern. It also illustrates a very important concept. There is no such thing as the correct program. The only test that can be applied is "Does the program work?" Certainly some programs are cleverer or accomplish the results more efficiently than others, but that is a separate issue.

Now back to the problem at hand. One way to approach the problem is to make the computer print out the first number in the pattern. Also, it is desirable to organize the program so that only a single print statement is required. Therefore, the computer should print out the value of a variable and change that value as it executes the program. Start the program with the following segment:

```
100 LET X=2
110 PRINT X,
```

The value of X is set to 2, and this value is printed in line 110. The comma causes the computer to space across to the next standard printing position. Now you need to generate the next value to be printed out. (Note that at any point in the number pattern, the next number is just 1 more than the present number.) You can generate the next value with the statement

```
120 LET X=X+1
```

Now all that remains is to program the computer to make the decision whether to loop back to the print statement or to stop. As long as X is less than or equal to 11, you want to loop back. You can do this with a conditional transfer statement.

```
130 IF X<=11 THEN 110
```

The program is finished with an END statement.

```
140 END
```

The complete program is

```
100 LET X=2
110 PRINT X,
120 LET X=X+1
130 IF X<=11 THEN 110
140 END
```

This program has little practical value but illustrates how a conditional transfer statement can get you out of a program at the proper time.

### Example 2 - Automobile License Fees

Assume that in an attempt to force consumers to use lower horsepower cars and thus conserve energy, the state adopts a set of progressive annual license fees based on the horsepower of cars. The criteria and fees are listed below.

Horsepower	License Fee
50 hp or less	\$0
More than 50 but 100 hp or less	30
More than 100 but 200 hp or less	70
More than 200 but 300 hp or less	150
More than 300 hp	500

Now think about a program that will produce the following output when executed:

```
INPUT AUTO HP? (You type horsepower)
LICENSE FEE IS (Computer types fee)
```

```
INPUT AUTO HP? (You type horsepower)
LICENSE FEE IS (Computer types fee)
```

(etc.)



Clearly, the only difficulty is programming the computer to decide what the fee is. This decision-making process is made to order for the IF THEN statement.

As a start, provide for input of the power rating. Use P to stand for the horsepower of the car. The program can begin with

```
100 PRINT "INPUT AUTO HP ";
110 INPUT P
```

Now, work out a method to decide in which license category P lies. A logical method is to check upward from the low horsepower ratings. First, the computer checks whether P is 50 or less. If so, then the tax is 0.

```
120 IF P<=50 THEN _____(Fee is 0)
```

The line number following THEN is missing for a reason. If the number in P is less than or equal to 50, the computer should jump to a statement that will assign the value 0 to the fee. But at this point you don't know what line number to use for this statement. Consequently, leave it blank and return later and insert the proper value. The note at the right reminds you what the fee is supposed to be if the assertion is true and the branch is taken.

If the assertion in line 120 is false, the computer will go to the next higher line number. In that line you want the computer to check whether P falls in the next higher category.

```
130 IF P<=100 THEN _____(Fee is $30)
```

Again, you don't know what line number to use following the THEN but can fill it in later.

Three further branch statements provide for every possible category of P. They are

```
140 IF P<=200 THEN _____(Fee is $70)
150 IF P<=300 THEN _____(Fee is $150)
160 IF P>300 THEN _____(Fee is $500)
```

The program to this point is

```
100 PRINT "INPUT AUTO HP ";
110 INPUT P
```

```

120 IF P<=50 THEN _____(Fee is 0)
130 IF P<=100 THEN _____(Fee is $30)
140 IF P<=200 THEN _____(Fee is $70)
150 IF P<=300 THEN _____(Fee is $150)
160 IF P>300 THEN _____(Fee is $500)

```

Now you can fill in the missing line number in line 120. Since the next line number in the program would be 170, use that number.

```

100 PRINT "INPUT AUTO HP ";
110 INPUT P
120 IF P<=50 THEN 170
130 IF P<=100 THEN _____(Fee is $30)
140 IF P<=200 THEN _____(Fee is $70)
150 IF P<=300 THEN _____(Fee is $150)
160 IF P>300 THEN _____(Fee is $500)
170 LET F=0
180 GOTO _____(PRINT statement)

```

Again, line 180 has a missing line number. Include a reminder that you want the computer to transfer to a PRINT statement. If the assertion in line 120 is true, the computer jumps to line 170 and assigns the value 0 to F, which stands for the fee. You can fill in the missing numbers in lines 130, 140, 150, and 160 using the same pattern. The result is

```

100 PRINT "INPUT AUTO HP ";
110 INPUT P
120 IF P<=50 THEN 170
130 IF P<=100 THEN 190
140 IF P<=200 THEN 210
150 IF P<=300 THEN 230
160 IF P>300 THEN 250
170 LET F=0
180 GOTO _____(PRINT statement)
190 LET F=30
200 GOTO _____(PRINT statement)
210 LET F=70
220 GOTO _____(PRINT statement)
230 LET F=150
240 GOTO _____(PRINT statement)
250 LET F=500

```

The next line in the program would be 260, which you can use for the first PRINT statement. The rest of the program follows easily. The complete program is

```
100 PRINT "INPUT AUTO HP ";
110 INPUT P
120 IF P<=50 THEN 170
130 IF P<=100 THEN 190
140 IF P<=200 THEN 210
150 IF P<=300 THEN 230
160 IF P>300 THEN 250
170 LET F=0
180 GOTO 260
190 LET F=30
200 GOTO 260
210 LET F=70
220 GOTO 260
230 LET F=150
240 GOTO 260
250 LET F=500
260 PRINT "LICENSE FEE IS ";F
270 PRINT
280 GOTO 100
290 END
```

You may have noticed that the conditional transfer statement in line 160 is not necessary. To see why, consider each of the assertions in the IF THEN statements. If the assertion in line 120 is false ( $P$  is greater than 50), the computer goes to the next higher line number, 130. Likewise, if each of the following assertions is false, the computer goes to the next higher line number. In particular, suppose the computer reaches line 150 and determines that the assertion is false. This directs the computer to line 160, but then  $P$  must be greater than 300, and therefore the computer can print the fee without testing further. If you assign the license fee of \$500 in line 160, the result is a slightly different program:

```
100 PRINT "INPUT AUTO HP ";
110 INPUT P
120 IF P<=50 THEN 200
130 IF P<=100 THEN 220
140 IF P<=200 THEN 240
```

```
150 IF P<=300 THEN 260
160 LET F=500
170 PRINT "LICENSE FEE IS ";F
180 PRINT
190 GOTO 100
200 LET F=0
210 GOTO 170
220 LET F=30
230 GOTO 170
240 LET F=70
250 GOTO 170
260 LET F=150
270 GOTO 170
280 END
```

Because the actions to be performed are short, another version of the program would be

```
100 PRINT "INPUT AUTO HP ";
110 INPUT P
120 IF P<=50 THEN F=0: GOTO 170
130 IF P<=100 THEN F=30: GOTO 170
140 IF P<=200 THEN F=70: GOTO 170
150 IF P<=300 THEN F=150: GOTO 170
160 LET F=500
170 PRINT "LICENSE FEE IS ";F
180 PRINT
190 GOTO 100
200 END
```

Notice the colon in line 120. The colon can be used to separate multiple statements after the THEN in an IF THEN statement.

These three versions of the program will work equally well, and you may have another version. You can decide how you prefer to handle the branches. The only test is whether your program works.

Study the program until you are convinced that it accomplishes what was desired. Also, remember to use the technique of leaving blanks in place of line numbers when you don't yet know what the line numbers should be, then returning later to fill in the proper lines. Write comments at the right of the blanks to help you

remember what you want to happen at that branch point in the program.

### Example 3 - Averaging Numbers

Suppose a DATA statement contains numbers that you wish to average. One problem is that you don't know in advance how many numbers there will be. To solve the problem use a flag variable to mark the end of the data. Make the flag a number that is very unlikely to occur in the data, for instance, the number 9999.

Here is the way it will work. The DATA statement will always appear as follows:

Line# DATA (number),(number),....,(number),9999

Each time the computer reads a number from the DATA statement, it checks to see if it is 9999. If not, the number is part of the data to be averaged. If the number is the flag, 9999, the computer has read all the data and can go on to the rest of the program.

The average is the sum of the numbers in the DATA statement divided by the number of numbers. Therefore, the program must give a way to compute both these quantities. Use S to stand for the sum of the numbers and N for the number of numbers. When the program is executed, the computer does not know what these values will be, so it sets them equal to 0 and develops their values as it reads numbers from the DATA statements.

Begin the program by defining the initial values of S and N, which in this case are both zero.

```
100 LET S=0
110 LET N=0
```

Now program the computer to read a number from the DATA statement and check for the flag value.

```
120 READ X
130 IF X=9999 THEN _____(Compute average)
```

Leave a blank in the conditional transfer statement until you know what line it should appear in. In this case, if the assertion ( $X = 9999$ ) is true, then all the numbers in the DATA statement have

been processed and the average can be computed. If the assertion is false, then the number just read is part of the data and should be processed. This is done as follows:

```
140 LET S=S+X
150 LET N=N+1
```

In line 140, the value of *X* (the number just read) is added to the value in *S*. Remember that the sum of all the numbers to be averaged is being developed in *S*. In line 150, 1 is added to *N* to record the fact that another number has been processed. Then add a GOTO to the READ statement, which instructs the computer to check the next number in DATA.

```
160 GOTO 120
```

Now you can fill in the missing number in line 130, since the next line number in the program would normally be 170. Line 170 computes the average, *A*. If a typical DATA statement is included, the complete program is

```
100 LET S=0
110 LET N=0
120 READ X
130 IF X=9999 THEN 170
140 LET S=S+X
150 LET N=N+1
160 GOTO 120
170 LET A=S/N
180 PRINT A
190 DATA 4,2,3,6,5,9999
200 END
```

Of course, you can use as many DATA statements as you need to accommodate the numbers to be averaged. After the last number in the last DATA statement, put the flag 9999 to mark the end of the data. The flag signals the computer to exit the READ loop and compute the average.

Remember the flag variable. Used with conditional statements, it will expand your programming capabilities.

## 5-5 FINDING ERRORS IN PROGRAMS

The ability to look at a program and determine whether it will accomplish what it is supposed to is certainly one of the most important skills you can acquire. Probably more to the point, you need the ability to find out what is wrong and correct it when a program is not working as it should. Although the task seems difficult, it is really easy.

Two separate skills are involved in troubleshooting programs. First, you need to decide for which variables you need additional information. You can insert PRINT statements into the program to have the computer print the value of the variables you want to see. Second, you need to be able to follow the logic of the program by going through the program as the computer would (using paper and pencil, if necessary). For loops, it is generally sufficient to check the first couple of values and the last couple of values. These two abilities together allow you to find logical programming errors quickly.

## 5-6 PROBLEMS

1. Write a BASIC program that calls for the input of two numbers and prints the larger.
2. Write a BASIC program to READ three numbers from a DATA statement and print the smallest.
3. Write a program to compute and print the sum of all the whole numbers between 1 and 100.
4. What will happen if the following program is executed? Describe in your own words.

```
100 LET S=0
110 LET X=1
120 LET S=S+X
130 LET X=X+2
140 IF X<100 THEN 120
150 PRINT S
160 END
```

5. Edit line 190 of example 3 (Averaging Numbers) as follows:

```
190 DATA 4,2,3,6,5,1111
```

Study the program and write down what you think the output will be when you execute the program. Run the program to see if you are correct. It may be helpful to insert line 155 as follows:

```
155 PRINT "S = ";S;" N = ";N
```

6. Write a program to find the average of all the positive numbers in a list whose end is marked with the flag 999. The list of numbers is to be typed in at RUN time.
7. Suppose you are given a DATA statement that contains a list of numbers. The end of the list is marked with the flag variable 9999. Write a BASIC program to compute and print the sum of the numbers in the list between -10 and +10 inclusive.
8. Usually supermarkets markup items depending on the unit cost of the item. Suppose this markup is based on the following schedule:

Unit Cost	Markup
0 to \$1.00	20%
\$1.01 to \$2.00	10%
over \$2.00	5%

The unit cost is determined by dividing the case price by the number of items in the case. Write a program to compute label price, which is unit cost plus markup. Arrange for prompts and input any way you desire.

9. Suppose you agree to work for one cent the first day, two cents the second, four cents the third, eight cents the fourth, and so on. If there are 22 working days in a month, write a program that will compute your wages (in dollars) for one month.



10. Consider the series

$$1 + 1/2 + 1/3 + 1/4 + \dots$$

Write a program to find the sum of the first N terms. Use the program to find the sum of the first 10, 100, and 1000 terms. Study the results. What do you think the sum would become as the number of terms increases forever.

11. Study the following program. Can you describe what the program does? You may wish to use the troubleshooting techniques discussed previously.

```
100 READ N
110 LET L=1
120 LET C=1
130 READ X
140 LET C=C + 1
150 IF X<L THEN 170
160 LET L=X
170 IF C<N THEN 130
180 PRINT L
190 DATA 10
200 DATA 5,83,17,3,47
210 DATA 25,16,41,51,7
220 END
```

You can obtain additional information on how the program works by inserting line 165 as follows

```
165 PRINT "L IS ";L
```

and running the program.

12. The following program is intended to find the average of N numbers typed in at the terminal. As it stands the program is incorrect. What's wrong?

```
100 PRINT "HOW MANY NUMBERS"
110 INPUT N
120 LET S=0
130 LET C=1
140 PRINT "TYPE IN A NUMBER";
```

```

150 INPUT X
160 LET S=S + X
170 LET C=C + 1
180 IF C<N THEN 140
190 LET A=S/N
200 PRINT "THE AVERAGE IS ";A
210 END

```

13. You can compute the discounted price of an item with the equation

$$D = L*(1 - R/100)$$

where L is the purchase price and R is the discount rate in percent. Write a program that will produce the following output when it is executed:

```

LIST PRICE ($)? (You input price)
DISCOUNT RATE (%)? (You input rate)
DISCOUNTED PRICE IS
(Computer displays price) DOLLARS

```

14. There is an interesting sequence of numbers called the Fibonacci numbers. The set begins with 0, 1. Each succeeding number in the sequence is the sum of the two previous ones. Thus, the Fibonacci sequence is

0, 1, 1, 2, 3, 5, 8, . . .

Write a BASIC program to compute and print the first 20 numbers in the Fibonacci sequence.

15. Write a program to accept the input of two numbers. If both the numbers are greater than or equal to 10, the computer should print their sum. If both the numbers are less than 10, the computer should print their product. If one number is greater than or equal to 10 and the other is less than 10, the computer should print the difference between the larger and smaller.

16. An instructor decides to award letter grades on an examination as follows:

90–100	A
80–89	B
60–79	C
50–59	D
0–49	F

Write a program to produce the following output when executed:

```
INPUT EXAM GRADE ? (You input numerical grade)
YOUR GRADE IS (Computer displays A, B, C, D, E, or F)
```

17. If you use 8 percent more electricity each year, in nine years your consumption will double. Thus, your doubling time is nine years. There is an interesting rule, called the rule of 72, that you can use to compute doubling times. If a quantity grows by  $R$  percent in a single period of time, then the number of periods for the quantity to double is given approximately by  $72/R$ . This is the rule of 72. You can compute the growth of a process directly on the computer. In a single growth period, a quantity  $Q$  grows according to the relation

$$Q_{new} = Q_{old}(1 + R/100)$$

Thus you can keep track of the growth by repeated use of the relation above. When  $Q$  is twice the original value, the corresponding number of growth periods would be the doubling time. Using this approach, write a program that will produce the following output when executed:

```
GROWTH RATE (%)? (You input R)
NUMBER OF GROWTH PERIODS TO DOUBLE IS
(Computer displays answer)
```

Use the program to check the accuracy of the rule of 72 for many different growth rates.

18. A set of integers (whole numbers) is chosen at random from the set 1, 2, 3, 4 and put in a DATA statement. The end of the set is marked with the flag 9999. Write a BASIC program that will compute and print out the number of 1s, 2s, 3s, and 4s in the set. Test your program on the following DATA statement:

```
DATA 3,1,2,1,4,4,1,2,2,2,3,9999
```

### 5-7 PRACTICE TEST

1. What will be the output if you run the following program?

```
100 LET Y=3
110 LET X=2*Y
120 PRINT X
130 LET Y=Y+2
140 IF Y<=10 THEN 110
150 END
```

---

2. What will be the output if you run the following program?

```
100 READ X
110 DATA 1,2,3
120 IF X<2 THEN 160
130 IF X=2 THEN 150
140 PRINT "GOOD"
150 PRINT "BETTER"
160 PRINT "BEST"
170 PRINT
180 GOTO 100
190 END
```

---

3. Suppose you decide to buy a number of widgets. The manufacturer is pushing sales and will give reduced prices for quantity purchases. The pricing schedule is

#Purchased	Price per Widget
20 or less	\$2.00
21 to 50	1.80
51 or more	1.50

Write a program that will produce the following output when executed and then loop back for new input.

```
HOW MANY WIDGETS ? (You input purchase quantity)
PRICE PER WIDGET IS (Computer displays unit price)
TOTAL COST OF ORDER IS (Computer displays total)
```

---

4. Write a program that will print out the number pattern shown below and then stop.

```
0  5  10 15 20 25
30 35 40 45 50 55
```

(etc.)

```
150 155 160 165 170 175
```

---

5. If you get a ticket for speeding, your fine is based on the number of miles per hour by which you exceeded the speed limit. Suppose the fine is computed as follows:

Amount over Limit	Fine
1-10 mph	\$ 5
11-20	10
21-30	20
31-40	40
41 or more	80

Write a BASIC program that will produce the following output when executed:

```
WHAT WAS SPEED LIMIT ? (You input limit)
SPEED ARRESTED AT ? (You input speed)
FINE IS (Computer displays fine) DOLLARS
```

---

# LOOPING AND FUNCTIONS

## 6—1 OBJECTIVES

### **Understanding Built-in Looping**

You have already learned how to loop programs by using either the unconditional or conditional transfer statements. Commodore 64 BASIC has special statements that cause automatic looping. These statements simplify the programming task and provide flexibility in programs.

### **Using Built-in Functions**

BASIC contains built-in functions that you can call on to perform specific tasks. You will look at some simple built-in functions that involve numerical computations and learn to use them to advantage.

### **Working with Program Examples**

You will continue activities that expand your programming ability. Remember that the overall objective of the book is to teach you how to write BASIC language programs.

## 6—2 DISCOVERY EXERCISES

1. Turn on the TV set and the computer. Enter the following program:

```
100 LET Y=10
110 PRINT Y;
120 LET Y=Y+5
130 IF Y<=50 THEN 110
140 END
```

Study the program and then execute it. Record what happened.

---

Which statement in the program determines the difference in the numbers that were displayed?

---

2. Clear out the program in memory. Now enter the following program:

```
100 FOR Y=10 TO 50 STEP 5
110 PRINT Y;
120 NEXT Y
130 END
```

Execute the program and record what happened.

---

Compare the output to the output you saw in step 1.

3. Since the two programs you just executed produce the same output, it is reasonable to assume that the statements must be related in some way. Type

```
100 FOR Y=10 TO 50 STEP 10
```

Display the program in memory and study it. What do you think will happen if you run this program?

---

See if you were right. Run the program and record the results below.

---



4. Now type

```
100 FOR Y=0 TO 5 STEP 1
```

Display the program. What do you think this program will do?

---

Run the program and write the output below.

---

5. Now type

```
100 FOR Y=0 TO 5
```

Display the program. What do you think this program will do?

---

Run the program and record what happened.

---

Compare line 100 in the program you just executed to line 100 in step 4. If the difference between the numbers to be printed out is 1, is the STEP part of the statement necessary?

---

6. Try a different tactic. Type

```
100 FOR Y=20 TO 10 STEP -2
```

Display the program and study it. What do you think this program will do?

---

Run the program and record the output.

---

## 7. Now type

```
100 FOR Y=10 TO 20 STEP -2
```

Display the program. What do you think will happen now if you run the program?

---

Run the program and write down what happened.

---

Notice that the FOR NEXT loop is executed one time, since 10 is printed out. You have fallen into a trap in BASIC. What is the problem?

---

8. So far the step sizes in your statements have allowed you to reach and display the limit: the number after TO. Now type

```
100 FOR Y=2 TO 9 STEP 3
```

Display the program. Write what you think the computer will display.

---

Run the program and record what happened.

---

9. Now go on to some more complex uses of FOR NEXT statements. Clear the program in memory and enter the following program:

```
100 FOR X=1 TO 3
110 FOR Y=1 TO 4
120 PRINT X,Y
130 NEXT Y
140 NEXT X
150 END
```

Run the program and record the output.

---

10. Now type

```
100 FOR X=1 TO 2
```

Run this new program and record the output.

---

Compare the two number patterns you have just obtained. Can you see the connection between the patterns and the limits in the FOR NEXT statements?

---

11. Modify the program a bit more. Type

```
100 FOR X=1 TO 3
110 FOR Y=1 TO 2
```

Display this program and study it. What do you think will be output if it is executed?

---

Try it and see if you were right.

---

## 12. One more time. Type

```
100 FOR X=1 TO 2  
110 FOR Y=1 TO 2
```

List the program and write what you think the computer will display when it executes the program.

---

Run the program and record the results below.

---

Display the program you just executed. Imagine drawing a line from the line number of the FOR Y statement to the line number of the NEXT Y statement. Do the same thing for the FOR X and the NEXT X statements. Do these lines cross?

---

## 13. Now type

```
100 FOR Y=1 TO 2  
110 FOR X=1 TO 2
```

Display the program. What do you think this program will do?

---

Run the program and record what happened.

---

On the listing of this program, imagine drawing lines from the FOR Y to the NEXT Y line numbers just as you did in step 12. Do the same thing for the FOR X and the NEXT X statements. Do these lines cross? Compare with the same situation in step 12.

---

Does the "NEXT WITHOUT FOR" ERROR suggest a way to avoid getting into trouble when you use more than one FOR NEXT combination in a single program?

---

14. In an earlier chapter you experimented with the TAB function to get variable spacing in the output. Now that you have the FOR NEXT statements at your disposal, go back to the TAB function (see the discussion section in Chapter 4). Clear the program in memory and enter the following program:

```
100 FOR X=1 TO 5
110 PRINT TAB(X);
120 FOR Y=X TO 5
130 PRINT "Y";
140 NEXT Y
150 PRINT
160 NEXT X
170 END
```

Take a few moments to study the program. What output do you think the program will produce?

---

See if you were right. Run the program and record the output below.

---

15. Clear the program in memory. Enter the program below.

```
100 INPUT A
110 LET B=SQR(A)
120 PRINT B
130 GOTO 100
140 END
```

Run the program and at the INPUT prompt, type 4. What happened?

---

Now type 9 and record the results.

---

One more time. Type 25. What happened?

---

Finally, type 10. What happened?

---

What happens to A in the expression SQR(A) in line 110 of the program? In other words, what does SQR do?

---

16. Jump the computer out of the input loop. Type

```
110 LET B=INT(A)
```

Run the program for the following values of A. In each case, record the output of the program.

A	Output
1	_____
3.4	_____
256.78	_____
0	_____
-1	_____
-2.3	_____

Examine the output you recorded above and compare each number with the corresponding value of A. What does the INT(A) function do?

---

If you had trouble understanding what happened to the negative values of A, don't worry at this point. We will review this completely later.

17. Jump the computer out of the input loop. Type

```
110 LET B=SGN(A)
```

Display the program and review the program structure to refresh your memory about how the program works. Run the program and input the following values of A. In each case, record the output.

A	Output
1.5	_____
43	_____
128.3	_____
0	_____
-1	_____
-1.2	_____
-345.7	_____
4.7	_____
-5.8	_____

Examine the output carefully. What does the SGN function do?

---

18. One more function. Jump the computer out of the input loop. Type

```
110 LET B=ABS(A)
```

Run the program and input the values of A given below. Again, record the output.

A	Output
3.4	_____
0	_____
-3.4	_____
-2	_____
-8.45	_____
8.45	_____

Examine the output. What does the ABS function do?

Interrupt the program.

19. This concludes the computer work for now. Turn off the computer and the TV set and go on to the next section.

## 6-3 DISCUSSION

### Understanding Built-in Looping

In the previous chapters you learned how to loop programs by using transfer statements. The unconditional (GOTO) statement was useful but could sometimes create a loop from which there was no exit. The conditional (IF THEN) statements gave you a way to loop programs and also a way to get out of the loop. All of these are good techniques. However, Commodore 64 BASIC has a very elegant looping technique that simplifies the programmer's task. This technique involves the use of FOR NEXT statements.

All FOR statements have the same format. This format and a typical statement are shown below.

Line# **FOR**(variable) = (relation)**TO**(relation) **STEP**(relation)

120 FOR X=1 TO 9 STEP 2



In FOR statements, the variable and the three relations can change. If you leave the STEP out of the statement, the computer will use a step size of 1. You can write the FOR statement in many different forms. A few are given below to illustrate the range of possibilities.

```
130 FOR J=2 TO 8
130 FOR T=25 TO 10 STEP -2
130 FOR W=-20 TO 10 STEP 2
130 FOR X=3*Z TO A*B STEP D
```

In general, you can write any legal BASIC statement as a relation in a FOR statement provided, of course, that you define the variables in the program.

The FOR statement opens a loop. The NEXT statement closes the loop.

```
200 FOR X=2 TO 18 STEP 2 (Opens loop)
.
.
.
Program lines inside loop
.
.
.
340 NEXT X (Closes loop)
```

The variable in the NEXT statement must be the same as the variable in the FOR statement.

- FOR opens a loop.
- NEXT closes that loop.

It is important to understand completely how these loops work. In the example above, when the program reaches line 200 the first time, the computer sets X equal to 2. Then the computer works through the lines until it reaches line 340. Line 340 closes the loop and directs the computer back to line 200 and the next value of X, in this case, 4. The computer stays in the loop until the value of X exceeds the limit of 18. Then, instead of going through the statements inside the loop, the computer closes the loop by jumping

to the line number following the NEXT statement. Look at one more example of FOR NEXT statements in action.

```

100 LET A=1
110 FOR X=1 TO 6 STEP 2
120 LET A=2*A
130 PRINT A,X
140 NEXT X
150 END

```

There are only two variables (A and X) in this program. You can list the line numbers in the order the computer encounters them and the corresponding values of the variables as follows:

Line Number	A	X
100	1	
110	1	1
120	2	1
130	2	1
140	2	1
110	2	3
120	4	3
130	4	3
140	4	3
110	4	5
120	8	5
130	8	5
140	8	5
110	8	7*
150	**	

\* Jumps out of loop

\*\* Program stops

Study the sequence of line numbers and the corresponding values of A and X until you are certain you understand how the FOR NEXT statements control the loop. When you run the program, line 130 will printout

2	1
4	3
8	5

Quite often, programs require more complicated loop structures. The structure can be as complex as necessary provided the loops do not cross. The example below illustrates a segment of a program with crossed loops.

```

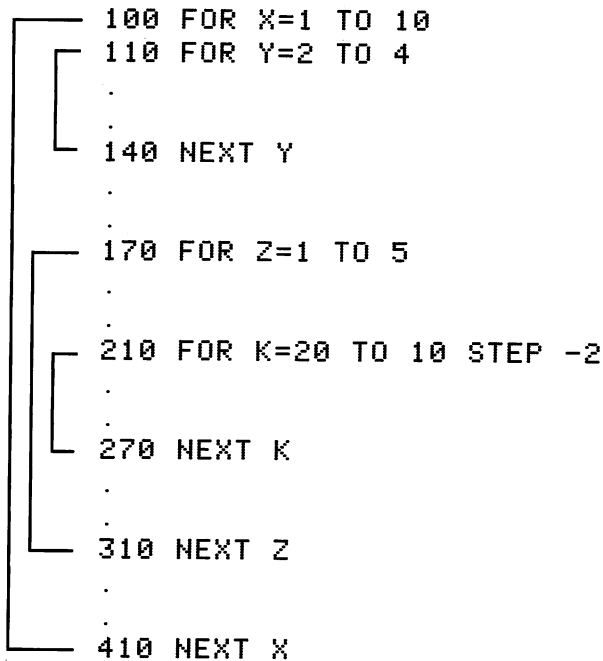
100 FOR A=2 TO 20
110 FOR B=4 TO 8
    Loops cross!
240 NEXT A
250 NEXT B
    
```

Another example with crossed loops is

```

100 FOR I=0 TO 20 STEP 2
110 FOR A=10 TO 2 STEP -1
120 FOR B=1 TO 4
    Outer loop OK; inner loops cross!
170 NEXT A
180 NEXT B
190 NEXT I
    
```

The following example illustrates a complicated loop structure in which the loops are organized correctly:



```
100 FOR X=1 TO 10
110 FOR Y=2 TO 4
.
.
140 NEXT Y
.
.
170 FOR Z=1 TO 5
.
.
210 FOR K=20 TO 10 STEP -2
.
.
270 NEXT K
.
.
310 NEXT Z
.
.
410 NEXT X
```

The diagram illustrates a nested loop structure. A large left bracket groups lines 100 through 410, indicating the outermost loop for X. Inside this, a bracket groups lines 110 through 140, indicating a loop for Y. Another bracket groups lines 170 through 310, indicating a loop for Z. A final bracket groups lines 210 through 270, indicating a loop for K. The lines connecting the FOR statements to their corresponding NEXT statements do not cross, showing a correct nesting of loops.

This example shows double loops and loops within loops. Remember, though, that any combination can be used in a program if the lines connecting the FOR statements and their corresponding NEXT statements do not cross. If they do, the computer will signal an error and stop.

### ■ Don't cross your FOR NEXT loops!

#### Using Built-in Functions

One advantage of modern digital computers is that sets of instructions can be preprogrammed to accomplish any desired task. Several built-in functions in BASIC allow the programmer to perform very complicated mathematical operations without difficulty. The table below gives some of these functions.

Function	Action
SQR(X)	Square root of X
INT(X)	Integer part of X
SGN(X)	Sign of X
ABS(X)	Absolute value of X

The first function, SQR(X), illustrates how all functions in general operate. X is called the argument of the function. If this definition bothers you, then think of X as “what the function works on.” If you use SQR(X) in a program, you are instructing the computer to look up the value of X, then take the square root of the number. For example,

$$\text{SQR}(36) = 6$$

$$\text{SQR}(64) = 8$$

$$\text{SQR}(81) = 9$$

$$\text{SQR}(2) = 1.41421356$$

and so on. A limitation of this function in particular is that you can't take the square root of a negative number. If you instruct the computer to evaluate SQR(−6), for example, it would signal an error and stop.

The argument of any function can be as complicated as it needs to be for the purposes of the program. If the expression is

$$\text{SQR}(X+4*Y)$$

the computer determines the values of X and Y, carries out the calculation of the argument indicated, then takes the square root if the calculated argument is positive. The computer simplifies the argument of all functions in the same way.

INT(X) takes the integer part of X. The term *integer* means “whole number.” Thus, 2 is an integer and 23.472 is not. If you take the integer part of a positive number, you simply forget about everything following the decimal point. Thus

$$\text{INT}(3.1593) = 3$$

$$\text{INT}(54.76) = 54$$

$$\text{INT}(0.362) = 0$$

However, negative numbers require special attention. What really happens when you take the integer part of a number is that you determine the first whole number less than or equal to the number. According to this rule, then,

$$\text{INT}(-2) = -2$$

$$\text{INT}(-.93) = -1$$

and so on. Note carefully that the INT function does not round off a number. Often beginners are somewhat confused about this.

- **The integer part of a number is the first whole number less than the number.**

SGN(X) is a very interesting function. If X (the argument) is positive, SGN(X) is +1. If X is negative, SGN(X) is -1. If X is 0, SGN(X) is 0. In effect, SGN(X) returns the sign of X, either +1, -1, or 0. Therefore,

$$\text{SGN}(4.568) = +1$$

$$\text{SGN}(375) = +1$$

$$\text{SGN}(0) = 0$$

$$\text{SGN}(-5.93) = -1$$

$$\text{SGN}(-4) = -1$$

At this point, it may not be clear how such a function could be useful. SGN is a very useful function, however, and has many applications. For the time being, just remember how the function works.

ABS(X) simply tells the computer to ignore the sign of X. In effect, it converts all values of X, other than 0, to positive numbers. So

$$\text{ABS}(4.5) = 4.5$$

$$\text{ABS}(-4.5) = 4.5$$

$$\text{ABS}(95.34) = 95.34$$

$$\text{ABS}(-95.34) = 95.34$$

$$\text{ABS}(0) = 0$$

BASIC has many other built-in functions. However, their study requires more mathematical knowledge than many students have. If you have the knowledge to understand what the functions instruct the computer to do, you will have no difficulty learning how to use them. If you are interested, consult the *Commodore 64 User's Guide*. We will take up some functions that involve strings of characters in the next chapter.

You can use built-in functions in BASIC statements. Examples of lines that use such functions might be

```
100 LET X=SQR(Y)
100 LET Z=3*INT(C)+ABS(D)
```

You can use built-in functions within other functions

```
100 LET Y=INT(SQR(X)+3*ABS(Z))
```

- Any BASIC expression can be the argument of a BASIC function.

## 6-4 WORKING WITH PROGRAM EXAMPLES

The following programs show how you can use automatic looping and the built-in functions to make programming easier.

### Example 1 - Finding the Average of a Group of Numbers

In the previous chapter, you solved the problem of finding an average for a group of numbers. Let's return to the same problem but use a different method. You want the program to make the computer display the following printout:

```
HOW MANY NUMBERS (You input how many)
ENTER THE NUMBERS, ONE AT A TIME
? (You enter the numbers)
THE AVERAGE IS (Computer displays the average)
```

The first few lines should be easy for you to write.

```
100 PRINT "HOW MANY NUMBERS";
110 INPUT N
120 PRINT "ENTER THE NUMBERS, ONE AT A TIME"
```

Now arrange for the input of  $N$  numbers but also keep in mind that you want the computer to determine the average of the numbers. So initially set  $S$  (the sum of the numbers) equal to 0.

```
130 LET S=0
```

FOR NEXT statements are ideal for the tasks of inputting  $N$  numbers and computing their sum.

```
140 FOR I=1 TO N
150 INPUT X
160 LET S=S+X
170 NEXT I
```

Notice that you use  $I$ , the loop variable, only to count the numbers as they are entered. When all the numbers are in, the computer jumps out of the loop to the next higher line number after 170. When this happens,  $S$  will contain the sum of all the values of  $X$  that were typed in. Since you know that  $N$  is the number of numbers typed in, you can immediately compute the average.

```
180 LET A=S/N
```

The rest of the program follows without difficulty.

```
190 PRINT "THE AVERAGE IS";A
200 END
```

The complete program is

```
100 PRINT "HOW MANY NUMBERS";
110 INPUT N
120 PRINT "ENTER THE NUMBERS, ONE AT A TIME"
130 LET S=0
140 FOR I=1 TO N
150 INPUT X
160 LET S=S+X
170 NEXT I
180 LET A=S/N
190 PRINT "THE AVERAGE IS";A
200 END
```



### Example 2 - Temperature Conversion Table

In an earlier program you used the relation

$$C = 5/9*(F-32)$$

to convert degrees Fahrenheit to degrees Celsius. Now let's generate a conversion table as follows:

Degrees F	Degrees C
0	-17.7777778
10	-12.2222222
20	-6.66666667
(etc.)	
100	37.77777

First write program lines that instruct the computer to print out the heading and the space before the table.

```
100 PRINT "DEGREES F", "DEGREES C"
110 PRINT
```

You can use a FOR NEXT loop to generate the values of F, which can then be converted to C and printed.

```
120 FOR F=0 TO 100 STEP 10
130 LET C=5*(F-32)/9
140 PRINT F,C
150 NEXT F
```

Finally, use the END statement.

```
160 END
```

The whole program is given below.

```
100 PRINT "DEGREES F", "DEGREES C"
110 PRINT
120 FOR F=0 TO 100 STEP 10
```

```
130 LET C=5*(F-32)/9
140 PRINT F,C
150 NEXT F
160 END
```

### Example 3 - Exact Division

The problem is to write a program that will compute all the integers (whole numbers) that divide exactly into another integer. For example, if 8 is the test integer, you want to find all the integers that will divide exactly into 8 with no remainder. The rule to use is

If  $N/X = \text{INT}(N/X)$  then there is no remainder

If  $N/X \neq \text{INT}(N/X)$  then there is a remainder

Now write a program to produce the following output when executed:

```
INPUT A POSITIVE WHOLE NUMBER? (Enter number)
NUMBERS THAT DIVIDE EXACTLY ARE
(Computer displays first number, second number, etc.)
```

The program begins easily.

```
100 PRINT "INPUT A POSITIVE WHOLE NUMBER";
110 INPUT N
120 PRINT "NUMBERS THAT DIVIDE";N;
    "EXACTLY ARE"
```

Now the program must instruct the computer to look at each of the whole numbers between 1 and N. Of course, this task is ideal for the FOR NEXT loops. Include the rule given above in the program to determine which numbers between 1 and N divide exactly into N.

```
130 FOR X=1 TO N
140 IF N/X <> INT(N/X) THEN 160
150 PRINT X,
160 NEXT X
```

Finally, use the END statement.

```
170 END
```

The complete program is

```
100 PRINT "INPUT A POSITIVE WHOLE NUMBER";
110 INPUT N
120 PRINT "NUMBERS THAT DIVIDE";N;
    "EXACTLY ARE"
130 FOR X=1 TO N
140 IF N/X <> INT(N/X) THEN 160
150 PRINT X,
160 NEXT X
170 END
```

You might test the program with fairly large values of N. How could you make the program run in half the time?

#### Example 4 - Depreciation Schedule

When a company invests in equipment, the investment is depreciated over a number of years for tax purposes. This means that the value of the equipment decreases each year (due to wear and tear), and the decrease is tax-deductible. One of the methods used to compute depreciation is the "sum-of-the-years'-digits" schedule. To illustrate, suppose a piece of equipment has a life of five years. The sum of the years' digits is

$$1 + 2 + 3 + 4 + 5 = 15$$

The depreciation the first year will be 5/15 of the initial value. The depreciation fraction the second year will be 4/15, and so on. If the equipment has an initial value of \$3000, the depreciation schedule is

End of Year	Depreciation Fraction	Depreciation	Current Value
1	5/15	1000	2000
2	4/15	800	1200
3	3/15	600	600
4	2/15	400	200
5	1/15	200	0

Your problem is to write a BASIC program to generate depreciation schedules by the "sum-of-the-years'-digits" method. The output should be as follows:

```
WHAT IS THE INITIAL ASSET VALUE? (You input value)
WHAT IS THE ASSET LIFE IN YEARS? (You input years)
END OF      DEPREC      DEPREC      CURRENT
YEAR        FRACTION
                                ASSET
                                VALUE
```

(Computer displays table)

The first few lines of the program require no explanation:

```
100 PRINT "WHAT IS THE INITIAL ASSET VALUE";
110 INPUT P
120 PRINT "WHAT IS THE ASSET LIFE IN YEARS";
130 INPUT N
140 PRINT
150 PRINT "END OF ", "DEPREC ",
"DEPREC ", "CURRENT"
160 PRINT " YEAR", "FRACTION"; TAB(30); "ASSET"
170 PRINT TAB(30); "VALUE"
```

Next, compute the "sum-of-the-years' digits."

```
180 LET S=0
190 FOR I=1 TO N
200 LET S=S+I
210 NEXT I
```

Now compute the schedule and print it. Use the variable P1 to keep track of the current asset value.

```

220 LET P1=P
230 FOR I=1 TO N
240 LET F=(N+1-I)/S
250 LET D=P*F
260 LET P1=P1-D
270 PRINT I,INT(F*100)/100,INT(D*100)/100,
  INT(P1*100)/100
280 NEXT I

```

In line 240, F is the depreciation fraction for the Ith year. You can check this out for various values of I to ensure that the expression does generate the correct value of F. In line 250, D is the depreciation. In line 270, the values of F, D, and P1 are rounded down to the nearest cent. A thorough discussion of a fairer rounding method is given in the first program example in the chapter on do-it-yourself functions. The only thing missing now is the END statement.

```
290 END
```

The complete program is

```

100 PRINT "WHAT IS THE INITIAL ASSET VALUE";
110 INPUT P
120 PRINT "WHAT IS THE ASSET LIFE IN YEARS";
130 INPUT N
140 PRINT
150 PRINT "END OF ","DEPREC ",
  "DEPREC ", "CURRENT"
160 PRINT " YEAR","FRACTION";TAB(30);"ASSET"
170 PRINT TAB(30);"VALUE"
180 LET S=0
190 FOR I=1 TO N
200 LET S=S+I
210 NEXT I
220 LET P1=P
230 FOR I=1 TO N
240 LET F=(N+1-I)/S
250 LET D=P*F

```

```

260 LET P1=P1-D
270 PRINT I,INT(F*100)/100,INT(D*100)/100,
  INT(P1*100)/100
280 NEXT I
290 END

```

Try the program with different inputs. Of course, you can use this to set up schedules for your tax returns. Impress the Internal Revenue Service with your computer-generated depreciation schedules!

## 6-5 PROBLEMS

1. Write a program to generate a table of numbers and their square roots. The table should look like the one below:

N	SQR(N)
2.0	1.414214356
2.2	1.4832397
2.4	1.5491933
(etc.)	

2. Write a program to count from 0 to 500 by 10's and print out the results.
3. Write a program to accept the input of a number N, then print out the even numbers greater than 0 but less than or equal to N.
4. Write a program to print out a conversion table from inches to centimeters. There are 2.54 centimeters in 1 inch. Include the appropriate headings. Start the table at 0 and continue to 10 inches in steps of 0.5 inch.
5. What will the computer display if you run the following program?

```

100 FOR X=10 TO 1 STEP -1
110 PRINT TAB(X); "ABCDEFGHIJ"
120 NEXT X
130 END

```

6. Study the following program. What will be the output?

```
100 FOR I=1 TO 5
110 READ A
120 LET B=INT(A)-SGN(A)*2
130 PRINT B
140 NEXT I
150 DATA 2,2,-3,10,0,-1.5
160 END
```

7. What will the computer print if you run the following program?

```
100 FOR X=1 TO 10
110 LET Y=2*X
120 FOR Z=1 TO 5
130 LET U=Z + Y
140 FOR V=1 TO 3
150 PRINT V + U
160 NEXT Z
170 NEXT V
180 NEXT X
190 END
```

8. The following program won't work. What's wrong?

```
100 FOR X=-10 TO +10 STEP 2
110 PRINT X, SQR(X)
120 NEXT X
130 END
```

9. Explain what the following program does:

```
100 FOR X=1 TO 5
110 READ Y
120 LET Z=INT(100*Y+.5)/100
130 PRINT Z
140 NEXT X
150 DATA 1.06142,27.5292,138.021
160 DATA .423715,51.9132
170 END
```

10. Write a program to print out the following pattern of asterisks:

```

* * * * *
  * * * *
    * * *
      *

```

Don't use more than three PRINT statements.

11. Explain what the following program does.

```

100 LET A=-4
110 LET B=ABS(A)
120 LET C=SQR(B)
130 LET D=SGN(A)+C
140 PRINT D
150 END

```

12.  $N!$  is read "N factorial" and means the product of all the counting numbers from 1 to N, inclusive. For example,

$$3! = (1)(2)(3) = 6$$

$$5! = (1)(2)(3)(4)(5) = 120$$

and so on. Write a program to call for the input of N. Then compute and print out  $N!$ . If you try this program on the computer, you may be surprised to find that there are values of N less than 100 that produce factorials too large for the computer to handle. The factorial of N is an extremely rapidly increasing function.

13. Write a BASIC program that calls for N grades to be input. Then program the computer to print (1) the highest grade, (2) the lowest grade, and (3) the average of the grades.
14. What, if anything, is wrong with the following program?

```

100 FOR X=1 TO 2
110 FOR Y=2 TO 6
120 PRINT X+Y
130 NEXT Y
140 FOR Z=1 TO 3
150 PRINT X+Z
160 NEXT X
170 NEXT Z
180 END

```



15. What will be the output if you run the following program?

```
100 FOR X=1 TO 4
110 FOR Y=1 TO 3
120 LET Z=X*Y
130 PRINT Z;
140 NEXT Y
150 PRINT
160 NEXT X
170 END
```

16. Suppose you decide to invest \$1000 on the first of each year for 10 years at an annual interest rate of 6 percent. At the end of the 10th year, the value of the investment will be \$13,971.64. To compute this, use the following formula:

$$\$NEW = (\$OLD + I)(1 + R/100).$$

In this formula, R is the annual interest rate in percentage. I is the annual investment, \$OLD is the value of the investment at the beginning of each year, and \$NEW is the value of the investment at the end of the year. Thus, \$NEW becomes \$OLD for the next year. Write a program that will produce the following output when executed:

```
ANNUAL INVESTMENT? (You enter I)
ANNUAL INTEREST RATE (%)? (You enter R)
HOW MANY YEARS? (You enter years)
AT THE END OF THE LAST YEAR THE VALUE OF THE
INVESTMENT WILL BE (Computer displays answer)
```

17. The DATA statements below contain the time worked by a number of employees during one week.

```
190 DATA 5
200 DATA 2, 4.8, 8, 10, 8, 7, 10
201 DATA 5, 3.75, 7, 8, 8, 6, 10
202 DATA 1, 3.25, 8, 10, 6, 8, 8
203 DATA 4, 5, 8, 10, 6, 10, 6
204 DATA 3, 4.25, 6, 6, 8, 10, 7
```

Line 190 gives the number of employees for whom DATA are entered. Each of the DATA lines after line 190 contains the

weekly record of one employee. The data are an employee number, the hourly rate, and the hours worked Monday through Friday. Employees receive time and a half for everything over 40 hours per week. Write a BASIC program using these DATA statements to compute and print the employee number and the gross weekly pay of each employee.

18. Assume that the following DATA statements give the performance of the students in an English class on three examinations:

```
190 DATA 6
200 DATA 3, 90, 85, 92
201 DATA 1, 75, 80, 71
202 DATA 6, 100, 82, 81
203 DATA 5, 40, 55, 43
204 DATA 2, 60, 71, 68
205 DATA 4, 38, 47, 42
```

Line 190 gives the number of students in the class. Each of the DATA statements that follow gives the performance of a single student. The information is student ID number, grade 1, grade 2, and grade 3. Thus, as shown in line 202, student 6 received examination grades of 100, 82, and 81. Write a program using these DATA statements to compute and print out each student's ID number and course grade. Assume that the first two examination grades are weighted 25 percent each toward the overall grade and the last grade is weighted 50 percent.

## 6-6 PRACTICE TEST

1. What will be printed if you run the following program?

```
100 FOR Y=20 TO 1 STEP -2
110 PRINT Y;
120 NEXT Y
130 END
```

---

2. What will be printed if the you run the following program?

```
100 FOR A=1 TO 4
110 FOR B=1 TO 3
120 PRINT A*B;
130 NEXT B
140 NEXT A
150 END
```

---

3. Fill in the blanks.

- a.  $\text{SQR}(36) = \underline{\hspace{2cm}}$
- b.  $\text{INT}(7.13) = \underline{\hspace{2cm}}$
- c.  $\text{ABS}(-22.8) = \underline{\hspace{2cm}}$
- d.  $\text{SGN}(-1.3) = \underline{\hspace{2cm}}$

4. What (if anything) is wrong with the following program?

```
100 FOR I=1 TO 5
110 FOR J=2 TO 5
120 PRINT I, J
130 NEXT I
140 NEXT J
150 END
```

---

5. You can convert miles to kilometers by multiplying the number of miles by 1.609. Write a program to produce the following output when executed.

MILES	KILOMETERS
-----	-----
10	16.09
20	32.18
30	
etc.	
100	160.9

- 
6. Two DATA statements contain the following numerical information.

```
100 DATA 10
110 DATA 25,21,24,21,26,27,25,24,23,24
```

Line 100 gives the number of numbers to be processed in the rest of the DATA statements. Write a program using these statements to compute the average of the numbers excluding the one in line 100.

---

# **WORKING WITH COLLECTIONS OF NUMBERS**

## **7-1 OBJECTIVES**

In this chapter you will apply some of the ideas you have learned earlier to collections of numbers. These new concepts will expand your ability to program in BASIC. The objectives are as follows.

### **Using Single- and Double-Subscripted Variables**

You will learn what subscripted variables are and how to use them to write powerful programs.

### **Saving Space for Arrays**

Before you enter a collection of numbers into the computer, the computer must know how large the collection will be. You will learn to use the DIM statement to define the size of the collection.

### **Using Subscripted Variables and FOR NEXT loops**

When you use a collection of numbers in a program, that program will almost certainly call for repetition. You will discover how FOR NEXT loops can help you handle the collection of numbers.

### **Working with Program Examples**

We will study BASIC programs that take advantage of the power of subscripted variables.

## 7-2 DISCOVERY EXERCISES

Since you will work with collections of numbers, you need to add two important words to your computer vocabulary. Although you could use the word *collection* to describe a group of numbers, two other words, *matrix* and *array* are more commonly used. In this book, they both mean the same thing: a "collection of numbers."

When you work with arrays, you need to be able to distinguish one number in the array from the other numbers. Subscripts help you distinguish the numbers in an array or matrix.

■ **MATRIX and ARRAY mean *collections of numbers*.**

To see how this works, look at the array given below.

$$Y_1 = 9$$

$$Y_2 = 10$$

$$Y_3 = 7$$

$$Y_4 = 14$$

$$Y_5 = 12$$

$$Y_6 = 15$$

The name of the array is Y. Its size is six, since there are six elements (or numbers) in it. The numbers 9, 10, 7, 14, 12, and 15 are the elements in the array. The numbers printed to the right and slightly below the Ys are called *subscripts*. Each subscript merely points to one element in the array. Thus, writing the subscript in parentheses, Y(4) means the fourth number in the array, which in this case is 14. Y(4) is read as "Y sub four," the third number in the array is "Y sub three," and so on. This array is one-dimensional, since it takes only a single subscript to locate a given element in the array.

Now look at a more complicated example:

$$Z_{1,1} = 4 \quad Z_{1,2} = 9 \quad Z_{1,3} = 5$$

$$Z_{2,1} = 3 \quad Z_{2,2} = 8 \quad Z_{2,3} = 7$$

In this example, there are six elements in the array Z. However, this array is two-dimensional, since you must specify an element in the array both by row and by column. The first subscript gives the row number; the second, the column.  $Z(2,1)$  is read as "Z sub two one" and means the element of Z at the second row and first column. Likewise, the element at row 1, column 3 would be identified as  $Z(1,3)$  ("Z sub one three").

In summary, you will work with two kinds of matrices or arrays. You need only one subscript to locate an element in a one-dimensional array, but two subscripts (a row number and a column number) to locate an element in a two-dimensional array.

1. Turn on the TV set and the computer. Enter the following program:

```
100 LET X(1)=21
110 LET X(2)=13
120 LET X(3)=16
130 LET X(4)=8
140 LET X(5)=11
150 PRINT X(1)
160 END
```

What do you think will be printed out if you run the program?

---

Run the program and record what happened.

---

2. Now modify the program to print out the fourth value of X. Run the program. Did it work?

---

3. Next, type

```
150 PRINT X(3) + X(4)
```

Display the program and study it briefly. What do you think will happen if you run the program?

---

Run the program and see if you were right. Record below what happened.

---

4. Type

```
150 FOR I=1 TO 5  
152 PRINT X(I)  
154 NEXT I
```

Display the program. What do you think this program instructs the computer to print.

---

See if you were right. Record below what happens when you run the program.

---

5. Modify this program to print out only the first three values of the array X. Record below what happened when you tried this.
- 

6. Again modify the program in line 150, but this time so that the first value of the array, then every other one, is printed. What changes and additions do you need to make to the FOR TO statement?
-



7. Clear out the program in memory. Enter the following program:

```
100 LET Y(1,1)=2
110 LET Y(1,2)=5
120 LET Y(1,3)=1
130 LET Y(2,1)=2
140 LET Y(2,2)=4
150 LET Y(2,3)=3
160 PRINT Y(1,3)
170 END
```

The array this program creates has the following rows and columns:

$$\begin{bmatrix} 2 & 5 & 1 \\ 2 & 4 & 3 \end{bmatrix}$$

Display the program and make sure you have entered it correctly.  
What do you think this program does?

---

Run the program and record what appeared on the screen.

---

8. Type

```
160 PRINT Y(2,2) + Y(1,3) + Y(1,1)
```

Display the program. What does this program do?

---

Run the program and see if you were right.

## 9. Type

```
160 LET S=0
162 FOR J=1 TO 3
164 LET S=S+Y(1,J)
166 NEXT J
168 PRINT S
```

Display the program and study it carefully. What will happen if you run this program?

---

Run the program and record what was printed out.

---

Explain in your own words what is taking place in the program.

---

## 10. Type

```
162 FOR I=1 TO 2
164 LET S=S+Y(I,2)
166 NEXT I
```

Display the program. What is the program doing now?

---

Run the program and write down what was printed out.

---

Again, explain in your own words what is happening.

---

## 11. Now type

```
164 FOR J=1 TO 3
166 LET S=S+Y(I,J)
168 NEXT J
170 NEXT I
172 PRINT S
180 END
```

Display the program and think a minute about it. In particular, compare what you see now to what was going on in steps 9 and 10. What does this program do?

---

Run the program and record what was typed out.

---

## 12. Clear the program in memory. Type the following program:

```
100 DIM X(12), Y(12)
110 FOR I=1 TO 12
120 READ X(I), Y(I)
130 NEXT I
140 PRINT X(1) + Y(4)
150 DATA 2, 1
151 DATA -1, 3
152 DATA 5, 6
153 DATA 2, 4
154 DATA 3, 1
155 DATA 8, 4
156 DATA 5, 1
157 DATA 3, 4
158 DATA 6, 2
159 DATA 1, 1
160 DATA 7, 7
161 DATA 5, 3
170 END
```

Display the program and check to see that you have entered it correctly. Study the program carefully. If you run the program, what will be typed out?

---

Run the program and see whether or not you were right. Record below what was typed out.

---

13. Type

100

Now display the first several lines of the program by typing LIST 100-150. What has happened?

---

Run the program and record what happened.

---

There is a BAD SUBSCRIPT larger than 10. Does the DIM statement that was originally in the program appear to be necessary?

---

14. Type the direct mode statement

PRINT I

What was the largest subscript (I) that the computer accepted?

---

## 15. Type

```
100 DIM X(9), Y(9)
110 FOR I=1 TO 9
```

Display the program. What will happen now if you execute the program?

---

Try it and see if you were correct.

## 16. Type

```
100
```

Will the program work now that the DIM statement has been taken out?

---

Try it and record the output.

---

Compare the results of step 13 with those of step 16. Sometimes the DIM statement must be present and other times it need not be. We will return to this question later.

## 17. Clear the program in memory. Type the following program:

```
100 DIM A(4,3)
110 FOR I=1 TO 4
120 FOR J=1 TO 3
130 READ A(I,J)
140 NEXT J
150 NEXT I
160 FOR I=1 TO 4
170 FOR J=1 TO 3
180 PRINT A(I,J);
190 NEXT J
200 PRINT
210 PRINT
220 NEXT I
```

```
230 DATA 1,3,1
240 DATA 4,2,5
250 DATA 1,4,2
260 DATA 3,2,5
270 END
```

Make sure that you have entered the program correctly, then take a few minutes to study it. Can you see what will be printed out if you run the program?

---

Run the program and record the output.

---

Compare what was printed out to the numbers in the DATA statements in the program.

18. Clear the program in memory and enter the following program:

```
100 DIM A(2,2)
110 FOR I=1 TO 2
120 INPUT A(I,1),A(I,2)
130 NEXT I
140 PRINT
150 PRINT
160 FOR I=1 TO 2
170 FOR J=1 TO 2
180 PRINT A(I,J);
190 NEXT J
200 PRINT
210 NEXT I
220 END
```

Run the program and when the INPUT prompt appears, type

```
2,5
3,8
```

What happened?

---

Compare the output to the numbers you typed in.

19. Clear the program in memory. Then enter the following program:

```
100 DIM X(3,3)
110 FOR I=1 TO 3
120 FOR J=1 TO 3
130 READ X(I,J)
140 NEXT J
150 NEXT I
160 PRINT
170 PRINT
180 FOR I=1 TO 3
190 FOR J=1 TO 3
200 PRINT X(I,J);
210 NEXT J
220 PRINT
230 NEXT I
240 DATA 2,1,3
250 DATA 4,7,5
260 DATA 1,2,6
270 END
```

Run the program. What happened?

---

Compare the output to the numbers in the DATA statements.

20. This concludes the computer work for this chapter. Turn off the computer and the TV set.

### 7-3 DISCUSSION

Usually most beginners are a bit confused about arrays at this point. The discussion material should clear up any questions that might have arisen in the computer work.

#### Using Single- and Double-Subscripted Variables

The need for subscripted numbers becomes obvious when you handle large collections of numbers. If, for example, you wrote a program

that involved only four numbers, you would have no difficulty naming them. You might call the numbers X, Y, U, and V. But suppose you needed to work with 100 numbers? For this reason, it is often very useful to have subscripted numbers. Fortunately, BASIC makes provisions for subscripts with which to name elements.

Consider the following set of numbers:

$i$	$Y_i$
1	14
2	8
3	9
4	11
5	16
6	20
7	5
8	3

You can refer to the entire set of numbers with the single name Y. Thus, Y is a collection of numbers, a matrix, or an array — all of which mean roughly the same thing, for the purpose of this book. To locate a number in an array, you must have the array name (in this case Y) and the number's position within the array. The  $i$  column gives a number's position. Thus  $Y(3)$  ("Y sub three") in the example above locates the third number in the array Y. In this case,  $Y(3)$  has the value 9. Similarly,  $Y(7)$  is 5,  $Y(1)$  is 14, and so on. You can assign the general name  $Y(I)$  ("Y sub I") to an as yet unspecified number in the array.  $Y(I)$  denotes any element of the array depending on the value of I. In the example above, if I were 8, then  $Y(I)$  would be 3. This collection of numbers is one dimensional, and you need only one number (subscript) to locate any element in the array.

Now look at a two-dimensional array.

$Y_{i,j}$	1	2	3	4
1	3	-1	10	8
2	2	4	5	6
3	1	-2	9	3



You need two numbers to locate an element in this array. Given a row number and a column number, you can find any element of the array. For example,  $Y(1,3)$  is the element of  $Y$  located at row 1, column 3. In the example above, the element has the value 10. A general way to denote an element in a two-dimensional array is  $Y(I,J)$ . The first subscript ( $I$ ) is the row number, and the second subscript ( $J$ ) is the column number.

■ **Double subscripts define row and column numbers.**

To make sure you understand how the double subscripts are used, refer to the two-dimensional array in the table above and verify that the following statements are correct:

$$Y_{3,2} = -2$$

$$Y_{1,4} = 8$$

$$Y_{3,3} = 9$$

$$Y_{2,1} = 2$$

The computer can interpret subscripts in the form of expressions. Thus,  $X(M-N+3, S*T)$  is legal provided that the computer can convert  $M-N+3$  and  $S*T$  into numbers. Even  $Y(Y(1,1), Y(2,3))$  is all right as long as the computer can locate the numbers in  $Y(1,1)$  and  $Y(2,3)$ . However, suppose you specify a number in an array as  $X(A+B)$ , and the computer searches the memory, where  $A$  is set equal to 2.6 and  $B$  to 1.1. Thus,  $A+B = 3.7$ . But it doesn't make any sense to try to look up the 3.7th number in the array  $X$ . Accordingly, the computer will take the integer part of 3.7 (or 3) and interpret  $X(A+B)$  as  $X(3)$ , the third element in the array  $X$ .

### **Saving Space for Arrays**

The computer must know how big an array is for two reasons. First, it must know how much space to save in memory to hold the array. Second, the computer must know the size of the array to carry out arithmetic operations properly. Actually, for small arrays, BASIC saves space automatically. If a one-dimensional array is used in a program, BASIC automatically sets up space for 10 elements if there

is no DIM statement. If a two-dimensional array is used, BASIC will save enough space in memory for a 10-by-10 array if no DIM statement is in the program. Even so, it is wise to use dimension statements in all programs regardless of the size of the arrays.

■ **Save space with a DIM statement.**

An example of a DIM (for “dimension”) statement is

```
100 DIM B(5,20),Y(8),Z(34),X(3,6)
```

Four arrays are dimensioned in this statement. **B** is a two-dimensional array having 5 rows and 20 columns. **Y** is a one-dimensional array with 8 elements. **Z** is one-dimensional array with 34 elements. Finally, **X** is a two-dimensional array with 3 rows and 6 columns. The DIM statement *must* precede any statements that refer to arrays. Make it a habit to make the DIM statement the first one of the program. That way you need only glance at the beginning of the program to see the sizes of the arrays that will be used.

**Using Subscripted Variables and FOR NEXT Loops.**

Since subscripts locate numbers in a collection, and since operations with collections of numbers almost always involve repetition, it is reasonable to use FOR NEXT statements to handle arrays. You can use FOR NEXT loops to define the subscripts used in the arrays. For example, the following program segment will set up a six-by-four array, then load 5s into all the elements.

```
100 DIM A(6,4)
110 FOR R=1 TO 6
120 FOR C=1 TO 4
130 LET A(R,C)=5
140 NEXT C
150 NEXT R
```

The details of the process become clear when you study this program segment. When line 130 in the program is reached the first time, **R** = 1 and **C** = 1. Then **R** is held constant while **C** goes to 2, 3, and 4. At each step in this process, the corresponding element of the array is set equal to 5. Then **R** is set equal to 2, and **C** takes on

the values 1, 2, 3, and 4. The process goes on until all the elements of the array have been set equal to 5.

You can use subscripts in this manner to handle either one- or two-dimensional arrays. In many applications, it is preferable to use FOR NEXT loops to carry out operations on arrays.

## 7-4 WORKING WITH PROGRAM EXAMPLES

### Example 1 - Examination Grades

To illustrate the concept of a one-dimensional array, we'll use a set of examination grades. The following are the results of an examination given to a class of 15 students:

	Student Number														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Grade	67	82	94	75	48	64	89	91	74	71	65	83	72	69	72

The problem is to write a BASIC program that allows you to enter the class grades. The format is:

```

HOW MANY STUDENTS? (You type in the number)
STUDENT GRADE
1 (You type first grade, etc.)
2
3
(etc.)

```

The program should compute the class average, find the highest grade and the lowest grade, and print this information as follows:

```

CLASS AVERAGE IS (Computer prints out average)
HIGHEST GRADE IS (Computer prints out highest grade)
LOWEST GRADE IS (Computer prints out lowest grade)

```

Take the problem by steps. First, since you are going to store the student grades in subscripted form, you must include a DIM statement to save space for the array.

```
100 DIM G(50)
```

Use the variable G to store grades. You can insert up to 50 grades. Next make provisions for a message, an input, and a space.

```
110 PRINT "HOW MANY STUDENTS";  
120 INPUT N  
130 PRINT
```

Generate the column heads for the table.

```
140 PRINT "STUDENT", "GRADE"  
150 PRINT
```

Now you are ready to enter the grades. A loop using FOR NEXT statements is ideal to control the input of grades.

```
160 FOR I=1 TO N  
170 PRINT I,  
180 INPUT G(I)  
190 NEXT I
```

Line 170 instructs the computer to print the student number. In line 180, the student number (I) is used as a subscript for the grade. This generates grades in the computer in the form G(1), G(2), . . . ,G(N).

The next task is to find the average of the grades. Do this with program lines that sum up all the grades and divide by the number of grades.

```
200 LET S=0  
210 FOR I=1 TO N  
220 LET S=S+G(I)  
230 NEXT I  
240 LET M=S/N
```

Now instruct the computer to print the results.

```
250 PRINT  
260 PRINT "CLASS AVERAGE IS";M
```

The final part of the program locates and prints the highest and lowest grades in the class. H and L will stand for the highest and lowest grades, respectively. Initially set both H and L equal to G(1), the first grade in the list. You know that the same grade can't be the highest and lowest at the same time (unless all the grades are the same). Thus, instruct the computer to go through the rest of the

grades, compare H and L with each grade, and make adjustments to H and L as required.

```
270 LET H=G(I)
280 LET L=G(I)
290 FOR I=2 TO N
300 IF L<G(I) THEN 320
310 LET L=G(I)
320 IF H>G(I) THEN 340
330 LET H=G(I)
340 NEXT I
```

Generate the required printout with these two lines.

```
350 PRINT "HIGHEST GRADE IS";H
360 PRINT "LOWEST GRADE IS";L
```

The END statement completes the program.

```
370 END
```

The complete program follows:

```
100 DIM G(50)
110 PRINT "HOW MANY STUDENTS";
120 INPUT N
130 PRINT
140 PRINT "STUDENT", "GRADE"
150 PRINT
160 FOR I=1 TO N
170 PRINT I,
180 INPUT G(I)
190 NEXT I
200 LET S=0
210 FOR I=1 TO N
220 LET S=S+G(I)
230 NEXT I
240 LET M=S/N
```

```

250 PRINT
260 PRINT "CLASS AVERAGE IS";M
270 LET H=G(1)
280 LET L=G(1)
290 FOR I=2 TO N
300 IF L<G(I) THEN 320
310 LET L=G(I)
320 IF H>G(I) THEN 340
330 LET H=G(I)
340 NEXT I
350 PRINT "HIGHEST GRADE IS";H
360 PRINT "LOWEST GRADE IS";L
370 END

```

Turn on the TV set and the computer and run this program using the data at the beginning of the discussion. If you have difficulty with the search for the highest and lowest grades (lines 270 through 340), trace the program in detail.

### Example 2 - Course Grades

You can easily extend the ideas in example 1 to a two-dimensional array. Suppose a class has 10 students, and the course grade is based on five examinations. Examination results for such a class might be

		Student Number									
		1	2	3	4	5	6	7	8	9	10
Exam	1	92	71	81	52	75	97	100	63	41	75
	2	85	63	79	49	71	91	93	58	52	71
	3	89	74	80	61	79	88	97	55	51	73
	4	96	68	84	58	80	93	95	61	47	70
	5	82	72	82	63	73	92	93	68	56	74

Use an array with the FOR NEXT statement to READ the data from DATA statements. The computer is to calculate and print the following:

STUDENT	COURSE AVERAGE
1	(Computer prints average, etc.)
2	
3	
(etc.)	
TEST	CLASS AVERAGE
1	(Computer prints average, etc.)
2	
3	
(etc.)	

The program should start with a DIM statement.

```
100 DIM G(5,10)
```

This statement reserves memory space for an array with 5 rows and 10 columns. The row number (R) will be the examination number, and the column number (C) will correspond to the student number. Put the DATA statements next, although they could appear anywhere in the program.

```
110 DATA 92,71,81,52,75,97,100,63,41,75
120 DATA 85,63,79,49,71,91,93,58,52,71
130 DATA 89,74,80,61,79,88,97,55,51,73
140 DATA 96,68,84,58,80,93,95,61,47,70
150 DATA 82,72,82,63,73,92,93,68,56,74
```

The following FOR NEXT statements, along with the READ statement in line 180, make all the data available to the computer:

```
160 FOR R=1 TO 5
170 FOR C=1 TO 10
180 READ G(R,C)
190 NEXT C
200 NEXT R
```

These statements cause the numbers to be read into the matrix G by rows. Thus, the data in line 110 become row 1 of the matrix G, and so forth. Next, use a PRINT statement to generate the required headings.

```
210 PRINT "STUDENT", "COURSE AVERAGE"  
220 PRINT
```

Now add statements to compute the course average for each student. First,

```
230 FOR C=1 TO 10
```

Line 230 opens a loop in which the computer will examine each column in the matrix. For each value of **C**, the computer will calculate the column average and print it.

```
240 LET S=0  
250 FOR R=1 TO 5  
260 LET S=S + G(R,C)  
270 NEXT R  
280 PRINT C,S/5
```

Then close the **C** loop.

```
290 NEXT C
```

Now repeat the process, but this time have the computer calculate the averages along rows rather than along columns. The results will reflect average class performance on each test.

```
300 PRINT  
310 PRINT "TEST", "CLASS AVERAGE"  
320 PRINT  
330 FOR R=1 TO 5  
340 LET S=0  
350 FOR C=1 TO 10  
360 LET S=S + G(R,C)  
370 NEXT C  
380 PRINT R,S/10  
390 NEXT R
```

Finally add an **END** statement.

```
400 END
```

The complete program follows:



```

100 DIM G(5,10)
110 DATA 92,71,81,52,75,97,100,63,41,75
120 DATA 85,63,79,49,71,91,93,58,52,71
130 DATA 89,74,80,61,79,88,97,55,51,73
140 DATA 96,68,84,58,80,93,95,61,47,70
150 DATA 82,72,82,63,73,92,93,68,56,74
160 FOR R=1 TO 5
170 FOR C=1 TO 10
180 READ G(R,C)
190 NEXT C
200 NEXT R
210 PRINT "STUDENT","COURSE AVERAGE"
220 PRINT
230 FOR C=1 TO 10
240 LET S=0
250 FOR R=1 TO 5
260 LET S=S + G(R,C)
270 NEXT R
280 PRINT C,S/5
290 NEXT C
300 PRINT
310 PRINT "TEST","CLASS AVERAGE"
320 PRINT
330 FOR R=1 TO 5
340 LET S=0
350 FOR C=1 TO 10
360 LET S=S + G(R,C)
370 NEXT C
380 PRINT R,S/10
390 NEXT R
400 END

```

This program illustrates valuable programming techniques involving arrays. It is worth studying and executing on your computer.

### Example 3 - Array Operations

A series of short programs presented without explanation follow. Study each program until you are sure you understand what is taking place.

- a. Write a program using FOR NEXT loops to load a three-by-four array with 1s.

```

100 DIM X(3,4)
110 FOR R=1 TO 3
120 FOR C=1 TO 4
130 LET X(R,C)=1
140 NEXT C
150 NEXT R
160 END

```

b. Write a program to generate and load the numbers

2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048

into a one-dimensional array.

```

100 DIM Z(11)
110 LET Z(1)=2
120 FOR I=2 TO 11
130 LET Z(I)=2*Z(I-1)
140 NEXT I
150 END

```

c. Write a program to read in the array

$$\begin{bmatrix} 2 & 3 & 5 \\ 1 & 4 & 2 \end{bmatrix}$$

from DATA statements and then print out the array.

```

100 DIM A(2,3)
110 FOR R=1 TO 2
120 FOR C=1 TO 3
130 READ A(R,C)
140 NEXT C
150 NEXT R
160 FOR R=1 TO 2
170 FOR C=1 TO 3
180 PRINT A(R,C);
190 NEXT C
200 PRINT
210 NEXT R
220 DATA 2,3,5
230 DATA 1,4,2
240 END

```

## 7-5 PROBLEMS

1. Write a program using the DATA statements

```
200 DATA 12
210 DATA 2,1,4,3,2,4,5,6,3,5,4,1
```

The program should read the size of an array from the first DATA statement, then read the elements of the array from the second DATA statement, load the elements into an array X, and print the array.

2. Write a program to fill a three-by-four array with 1s.
3. Write a program to call for the input of a square N-by-N matrix where N is a whole number no larger than 10. Then program the computer to calculate and print the sum of the entries on the main diagonal of the array (where the first and second subscripts are the same).
4. Write a BASIC program using the READ command to read 25 numbers from the DATA statements into a one-dimensional array named A. Include in your program instructions to search the array and print the number of elements in the array that are greater than 50. Fill in the required DATA statements with any numbers you choose.
5. Write a program to call for the input of an M-by-N matrix. Assume that both M and N are no larger than 15. Then program the computer to calculate and print the sum of all the elements in the matrix.
6. The program below is supposed to compute and print out the sum of the elements in a one-dimensional array that are positive but not greater than 10. As it stands the program is incorrect. What's wrong?

```
100 DIM A(6)
110 FOR I=1 TO 6
120 INPUT A(I)
130 NEXT I
140 LET S=0
150 FOR I=6 TO 1 STEP -1
160 IF A(I)>10 THEN 180
170 LET S=S+A(I)
180 NEXT I
```

```
190 PRINT S
200 END
```

7. What will be output if you run the following program?

```
100 DIM Y(6)
110 FOR I=1 TO 6
120 READ Y(I)
130 NEXT I
140 DATA 2,1,3,1,2,1
150 LET S1=0
160 LET S2=0
170 FOR I=1 TO 6
180 LET S1=S1 + Y(I)
190 LET S2=S2 + Y(I)^2
200 NEXT I
210 LET X=S2 - S1
220 PRINT X
230 END
```

8. What will be output if you run the following program?

```
100 DIM A(10)
110 FOR I=1 TO 10
120 READ A(I)
130 NEXT I
140 LET S=A(1)
150 FOR I=1 TO 9
160 LET A(I)=A(I+1)
170 NEXT I
180 LET A(10)=S
190 FOR I=1 TO 10
200 PRINT A(I)
210 NEXT I
220 DATA 10,9,8,7,6,5,4,3,2,1
230 END
```

9. What will be printed if you run the following program?

```
100 DIM X(4,4)
110 FOR I=1 TO 4
120 FOR J=1 TO 4
```

```
130 READ X(I,J)
140 NEXT J
150 NEXT I
160 DATA 1,2,3,4,2,3,4,5
170 DATA 3,4,5,6,4,5,6,7
180 LET S=0
190 FOR I=1 TO 4
200 LET S=S+X(I,5-I)
210 NEXT I
220 PRINT S
230 END
```

10. What will be printed if you run the following program?

```
100 DIM Y(4,4)
110 FOR R=1 TO 4
120 FOR C=1 TO 4
130 LET Y(R,C)=0
140 NEXT C
150 NEXT R
160 FOR R=1 TO 4
170 FOR C=1 TO 4
180 LET Y(R,C)=R*C
190 NEXT C
200 NEXT R
210 FOR R=1 TO 4
220 FOR C=1 TO 4
230 PRINT Y(R,C);
240 NEXT C
250 PRINT
260 NEXT R
270 END
```

11. Write a BASIC program to call for the input of N (assumed to be a whole number between 1 and 100), then input a one-dimensional array with N elements. The program should instruct the computer to sort the array into descending order and finally print the sorted array.
12. Assume that the first number in the DATA statements gives the number of pieces of data to follow. Also assume that the pieces of data are all whole numbers between 1 and 10 inclusive. Write a

program that will compute the number of 1s, 2s, etc., in the data and then print the results. (Hint: Use the data as they are read in as a subscript to increment an element of an array used to count the numbers.)

13. What will be printed if you run the following program?

```

100 DIM Z(6,6)
110 FOR R=1 TO 6
120 FOR C=1 TO 6
130 LET Z(R,C)=0
140 NEXT C
150 NEXT R
160 FOR R=1 TO 5 STEP 2
170 FOR C=R TO 6
180 LET Z(R,C)=1
190 NEXT C
200 NEXT R
210 FOR R=1 TO 6
220 FOR C=1 TO 6
230 PRINT Z(R,C);
240 NEXT C
250 PRINT
260 NEXT R
270 END

```

14. What will be output if you run the program below?

```

100 DIM A(5,5)
110 FOR R=1 TO 5
120 FOR C=1 TO 5
130 READ A(R,C)
140 NEXT C
150 NEXT R
160 DATA 2,2,2,2,2,2,2,2,2,2
170 DATA 2,2,2,2,2,2,2,2,2,2
180 DATA 2,2,2,2,2
190 FOR C=5 TO 1 STEP -1
200 FOR R=1 TO C
210 LET A(R,C)=3
220 NEXT R
230 NEXT C

```

```

240 FOR R=1 TO 5
250 FOR C=1 TO 5
260 PRINT A(R,C);
270 NEXT C
280 PRINT
290 NEXT R
300 END

```

15. Write a program to read the following array from DATA statements, then print the array.

$$\begin{bmatrix} 2 & 1 & 0 & 5 & 1 \\ 3 & 2 & 1 & 3 & 1 \end{bmatrix}$$

16. Write a program to read the following array from DATA statements, then print the array.

$$\begin{bmatrix} 5 & 3 \\ 2 & 0 \\ -1 & 1 \\ 4 & 2 \\ 2 & 6 \end{bmatrix}$$

17. Write a BASIC program that will call for the input of an M-by-N array, then compute and print out the sum of the elements in each row and the product of the elements in each column.
18. Write a BASIC program that will read two two-by-three arrays from DATA statements, then compute a third two-by-three array such that each element is the sum of the corresponding elements in the first two arrays. Finally, have the computer print the third array.
19. The data below represent the totals of sales made by four salespeople during one week.

	Mon	Tue	Wed	Thu	Fri	Sat
1	48	40	73	120	100	90
2	75	130	90	40	110	85
3	50	72	140	125	106	92
4	108	75	92	152	91	87

Write a program that will compute and print out

- a. The daily sales totals
- b. The weekly sales totals for each salesperson
- c. The total weekly sales

20. Write a program to call for the input of a four-by-four matrix, then compute a new matrix from the first with the rows and columns interchanged. That is, row 1 of the input matrix becomes column 1 of the new matrix. Row 2 of the input matrix becomes column 2 of the new matrix, and so on. Then have the computer print the new matrix.
21. Consider the two arrays below:

P	X
1	28
5	2
3	14
6	3
4	17
2	9

Each element of **P** "points" to an element of **X**.  $P(1) = 1$  and  $X(1) = 28$ .  $P(2) = 5$  and  $X(5) = 17$ . If you continue this process, the values of **X** are listed in descending order. Write a program to set up two arrays **X**, and **P**, to some convenient length. Then call for the input of arbitrary values of **X** from the keyboard. Construct the array **P** so that its elements point to **X** in descending order as illustrated above. Then print out the two arrays as shown and the arbitrary values of **X** in descending order.

## 7-6 PRACTICE TEST

1. What is the purpose of the DIM statement?  
\_\_\_\_\_
2. In the array named **X**, what variable name in BASIC locates the element in row 3, column 4?  
\_\_\_\_\_



3. Use an array in a program to input a list of numbers, then find and print out the sum of the positive numbers in the list. The printout should look as follows:

HOW MANY NUMBERS? (You type in the number)  
WHAT ARE THE NUMBERS? (You type them in)  
THE SUM OF POSITIVE ELEMENTS IS (Computer  
types out answer)

---

4. Write a program using FOR NEXT statements to load a four by six array with 4s.
- 

5. What will be printed if you run the following program?

```
100 DIM A(5,5)
110 FOR I=1 TO 5
120 FOR J=1 TO 5
130 LET A(I,J)=0
140 NEXT J
150 NEXT I
160 FOR I=1 TO 5
170 LET A(I,I)=2
180 NEXT I
190 FOR I=1 TO 5
200 FOR J=1 TO 5
210 PRINT A(I,J);
220 NEXT J
230 PRINT
240 NEXT I
250 END
```

---

6. The following array is named **A**:

$$\begin{bmatrix} 1 & 3 & 5 \\ 6 & 2 & 4 \end{bmatrix}$$

- a. Write a DIM statement for **A**.

---

- b. What is the value of **A(2,3)**?

---

- c. If  $X = 1$  and  $Y = 2$ , what is **A(X,Y)**?

---

- d. What is **A(A(1,1),A(2,2))**?

---

# STRING VARIABLES

## 8—1 OBJECTIVES

Some of the most important applications of computers are non-numeric and deal with characters rather than numbers. Strings of characters can be handled as “string variables” and are the subject of this chapter. Specifically you will study the following string-related topics.

### **Understanding String Input and Output**

Before you can perform meaningful operations with strings, you need to learn how the computer handles input and output of string variables.

### **Using String Functions**

You have already studied BASIC functions that operated on numbers. Now you will learn built-in functions that work on strings of characters.

### **Working with Program Examples**

The final goal is to write programs that use string variables.

## 8—2 DISCOVERY EXERCISES

1. Turn on the TV set and the computer. Enter the following program:

```
100 INPUT A$
110 PRINT
120 PRINT A$
130 PRINT A$
140 END
```

The A\$ in the program identifies the variable as a string variable. Run the program and at the input prompt (the question mark) type in your full name. What happened?

---

2. Now modify the program as follows:

```
100 INPUT A$,B$
110 PRINT
120 PRINT B$
130 PRINT A$
140 END
```

If you execute the program and at the input prompt type the words INTELLIGENT and CONVERSATION separated by a comma what do you think the computer will print?

---

Try it and record what happened.

---

3. Now try this variation. Clear the program in memory and enter the following:

```
100 READ X,X$,Y,Y$
110 DATA 10,HERB,20,CHARLIE
120 PRINT X,Y
130 PRINT X$,Y$
140 END
```

This program illustrates several new ideas. What do you think will happen if you run the program?

---

Run the program and record what took place.

---

4. As you see, you can include strings in DATA statements. Change line 110 as follows:

```
110 DATA 10,"HERB",20,"CHARLIE"
```

Run the program. What happened?

---

5. Change line 110 as follows:

```
110 DATA "10","HERB",20,CHARLIE
```

Display the program. Run the program. What happened?

---

"10" is a string while X is looking for a number.

6. Now change line 100 to read

```
100 READ X$,X,Y,Y$
```

What do you think will happen if you try to run the program in this form?

---

See if you were right. In the space below, record what happened when you tried to run the program.

---

7. Again, the syntax error is caused by a conflict in data types. Display the program and observe that HERB is a string, but X is a numeric variable. Clearly, you must be careful that the type of information in the DATA statements matches the type of variable in READ statements. Now on to a different topic. Clear the memory and enter the following:

```
100 INPUT C$
110 LET N=LEN(C$)
120 PRINT N
130 END
```

The new feature in this program is the function LEN. It works on a string (in this case the string is C\$) rather than a number. Can you guess what the function does?

---

Notice that the result of the function LEN operating on a string must be a number since the result is assigned to a numeric variable, N.

8. Run the program and at the input prompt type in ABCDE. What did the computer print?
- 

Try it again, but this time type in AARDVARK. What happened?

---

By now you should have a pretty good idea what the LEN function does.

9. Here's a new question. If you execute the program and at the input prompt merely press the RETURN key, what do you think the computer will print?
- 

Try it and see if you were correct. One more variation. Run the program and at the input prompt type in R O B E R T. Note the spaces between the characters. What was the output?

---

In the LEN function, do spaces count as characters?

---

10. Now explore how to specify a substring in a given string. That is, in the string A\$, how can you specify M characters of the string starting with the Nth character in the string? The function that yields this substring is MID\$(A\$,N,M). It gives M characters from A\$ starting at the Nth character. Clear the program in memory and enter the one below:

```
100 INPUT A$
110 PRINT "M = ";
120 INPUT M
130 PRINT "N = ";
140 INPUT N
150 PRINT MID$(A$,N,M)
160 END
```

Run the program and at the input prompt, type MISSISSIPPI. Note that the length of this string is 11. Enter 5 for M and 4 for N. What happened?

---

11. Clear the program in memory and enter the one below:

```
100 INPUT A$
110 INPUT J
120 PRINT MID$(A$,J,1)
130 PRINT
140 GOTO 110
150 END
```

Run the program and at the first input prompt type in ABCDEFGHIJKLMNOPQRSTUVWXYZ. At the second input prompt type in 20. What happened?

---

The computer is at the input prompt for J. This time, type in 10. What happened?

---

In response to the third input prompt, experiment with various values of I between 1 and 26. Describe in your own words what happens when the computer is directed to print out MID\$(A\$,J,1).

---

12. Check one last thing. You should still be at the input prompt for J. Type in 30. What happened?
- 

By now you should understand fairly clearly what happens when the computer prints out MID\$(A\$,J,1). Of course, when you entered 30, the value of J was greater than the length of the string. We will return to this topic in the discussion section. Jump your computer out of the input loop.

13. Experiment on your own with this program. Modify line 120 of the program. Try various strings and different values until you understand exactly how the MID\$ function works.
14. Now on to a different topic. Clear the program in memory and enter the following:

```
100 INPUT A$
110 INPUT B$
120 IF A$ < B$ THEN 150
130 PRINT B$
140 GOTO 100
150 PRINT A$
160 GOTO 100
170 END
```

Take a few moments to study the program. Clearly, the interesting statement is in line 120, in which the strings A\$ and B\$ are used in an IF THEN statement. What do you suppose A\$ < B\$ means with regard to strings?

---



Test your theory this way: Run the program and at the first input prompt, type in DUCK; at the second prompt, type in FISH. What was displayed?

---

15. The computer is at the input prompt waiting for you to enter a string. This time type in HOUSE followed by TELEVISION. Which was displayed?
- 

Keep experimenting with words or letters of your choice until you see exactly what the expression `A$ < B$` means. Once you understand this, you can guess what `A$ = B$`, `A$ >= B$`, or `A$ <> B$` mean.

16. Jump the computer out of the input loop. Clear the program in memory, and go on to the next string variable function: `CHR$`.

`CHR$` is a function that converts a number to a character on the keyboard. Most characters on the keyboard correspond to a number. Enter the following program:

```
100 INPUT N
110 PRINT CHR$(N)
120 GOTO 100
130 END
```

Now run the program and at the input prompt, type in 65. What happened?

---

The program keeps looping as long as you wish. This time try 66. What happened?

---

You may wish to try 49, 50, and 255 also.

17. Experiment with this program; try out various numerical inputs. Keep the numbers in the range 32 to 255. As you can see, you can refer to a character either by the character itself or by its position number in the set of characters. Interrupt the program.

18. To see a portion of the ASCII character set, clear the memory and type in and run the following program. The symbols that will appear on the screen are a of the few Commodore 64 graphics characters.

```
100 FOR N=65 TO 122
110 PRINT CHR$(N); " ";
120 NEXT N
130 END
```

You can see the entire ASCII character set on pages 135-137 of Appendix F of the *Commodore 64 User's Guide*. After running this program, holding down the Commodore Key (C=) at the far lower left of the keyboard) and then pressing the SHIFT key changes the characters on the screen. Repeating this operation changes the screen again. This is an easy way to see the differences between upper/lower case mode and upper case/graphics mode.

19. Now on to a different topic. First, clear the program in memory. ASC is the function that converts a character to its equivalent position number in the character set used by the computer. Type in the following program:

```
100 INPUT A$
110 LET X=ASC(A$)
120 PRINT X
130 GOTO 100
140 END
```

Run the program and at the input prompt, type Z. What happened?

---

This new function, ASC(A\$), is just the reverse of CHR\$(N). Use various letters and numbers and compare your results to those you obtained in step 16.

20. Check one last detail. Your computer should still be at the input prompt waiting for A\$ to be typed in. This time type in POTATO. Note the output. Then try PEA. What happened?
-

Notice that the two words begin with the same first letter. You will review this concept later.

21. Numbers can be considered strings of digits. Thus a number can be entered as a string of digits. VAL is the function that converts such strings of digits to their numeric value. Clear the program in memory and type in the following program.

```
100 INPUT A$,B$
110 PRINT A$ + B$
120 PRINT VAL(A$) + VAL(B$)
130 END
```

Display the program. Run the program. At the input prompt, type 5,6. Is the output from line 110 or 120 the arithmetic sum of 5 and 6?

---

Notice the extra space reserved for a minus sign before the 11.

22. The STR\$ function converts a number to its string of digits. Clear the program in memory and type in the following program.

```
100 INPUT A
110 LET A$=STR$(A)
120 PRINT MID$(A$,1,4)
130 END
```

Display the program. Run the program. At the input prompt, type 1234. Were all four digits typed out?

---

Here again, the first space is reserved for a minus sign. Thus, the string representing the number 1234 is five characters long.

23. In the previous chapter, we used numeric arrays. You can also define and use string arrays in BASIC programs. Type the following program.

```

100 DIM A$(6)
110 FOR I=1 TO 6
120 READ A$(I)
130 PRINT A$(I);
140 NEXT I
150 DATA 8,SUE,9,FRED,10,ANN
160 END

```

Run the program. Record what was printed out.

---

Here 8, 9, and 10 are strings of characters in A\$, an array of strings. To sum 8, 9, and 10 numerically, add line 145 as follows:

```

145 PRINT VAL(A$(1))+VAL(A$(3))+VAL(A$(5))

```

Run the program. Is the sum correct?

---

Change lines 100, 120, and 130 as follows.

```

100 DIM A(6)
120 READ A(I)
130 PRINT A(I);

```

Display the program. Run the program. Was a number printed?

---

What error message occurred?

---

24. This concludes the computer work for this chapter. Turn off TV set and the computer and go on to the next section.

### 8-3 DISCUSSION

#### Understanding String Input and Output

As you already know, a set of characters surrounded by quotation marks is called a string. The quotation marks are not part of the string, however. The new idea in this chapter is that the string can be treated as a variable — the string variable.

The string variable is identified by adding a dollar sign (\$) to the end of a name. **BARN\$, BOX\$, and B\$** are string variable names.

The computer handles input and output of string variables the same way as input and output of numeric variables. You can mix numeric and string variables in the same BASIC statements. Examples are

```
100 PRINT A$,X,Y,Z$
110 INPUT M$,N
120 READ A$,B$,Z
```

You must be careful that the input in either INPUT or READ statements matches the type of variable given. As it executes line 110 above, the computer expects a string of characters and a number. However, be aware that you can type in 123456789 and, if the computer expects a string, it will identify that quantity as a string, not as a number. The reason is that a string consists of characters, and the symbols 0 through 9 are part of the standard character set. If, however, the computer expects a number and you type in ABCDEFGHI, you will get an error message.

You can dimension and use string arrays in BASIC programs. Each element of a string array is a string of at most 255 characters. For example, if you run the following program

```
100 DIM A$(60)
110 LET A$(51)="12"
120 LET A$(52)="34"
130 PRINT A$(51) + A$(52)
140 PRINT VAL(A$(51)) + VAL(A$(52))
150 END
```

line 130 will print 1234, since + joins the two strings A\$(51) and A\$(52) together. Line 140 will print 46, the sum of the numbers 12 and 34. This example points up the difference between string and numeric variables and also between string and numeric arrays.

### Using String Functions

The LEN function is used to determine the length of a string. If, for example, A\$ = "HOW NOW BROWN COW", then LEN(A\$) = 17. Note that the spaces count as characters. You can also have a "null" string. If A\$ = "" (there is nothing inside the quotation marks), then LEN(A\$) = 0.

- **LEN(A\$) gives the number of characters in A\$.**

A substring is a piece or segment of a string. There are several ways to deal with substrings. Consider the following program:

```
100 LET A$="ROBERT E. LEE"
110 PRINT MID$(A$,8,6)
120 END
```

The expression MID\$(A\$,8,6) identifies the substring of A\$ consisting of six characters starting at the eighth character. If the program were run, the characters E. LEE would be printed out.

- **MID\$(A\$, I,J) gives J characters of string A\$ starting at character I.**

You can compare string variables in IF THEN statements. The comparison is alphabetical. Thus A < B since A comes before B in the alphabet. CAT < DOG, HOUSE > CAR, PEA < PEARL, and so on.

To understand string functions, you need to know about the ASCII (American Standard Code for Information Interchange) standard character set. This set is used on most computers and consists of 128 characters numbered 0 through 127. Refer to the *Commodore 64 User's Guide* for a complete listing of the ASCII character set.

In upper case/graphics mode, the uppercase letters A through Z are numbered 65 through 90. In the upper/lower case mode, the lowercase letters a through z correspond to the numbers 65 through 90. The numerals 0 through 9 are numbered 48 through 57. Other numbered characters include punctuation marks, arithmetic operators (+, -, \*, etc.) and other special and graphics characters.

Since there are two keyboard modes, each number may represent two screen symbols.

Two string functions work with the ASCII character set. First, `CHR$(N)` returns the Nth character from the ASCII character set. For example, `CHR$(65) = A`, `CHR$(90) = Z`, and so forth. You can also convert a character to its ASCII number with the `ASC(A$)` function. For example, `ASC("A") = 65` and `ASC("Z") = 90`.

Suppose that `A$ = "AIRPLANE"`. What is `ASC(A$)`? When the length of the string is greater than one, the computer considers only the first character. Since the first character is A, `ASC(A$) = 65`. Two functions are used in dealing with numbers as strings of digits. `STR$(-945)` returns a string whose characters are a minus sign, 9, 4, and 5. The function `VAL` returns the numeric value of a string whose characters are digits. Thus, `VAL("112233445566")` returns 1.12233446E+11.

## 8-4 WORKING WITH PROGRAM EXAMPLES

### Example 1 - String Reversal

The task is to write a program to call for the input of a string and then print it back in reverse order. First, arrange for the string input.

```
100 INPUT A$
```

The next few lines print the string in reverse order.

```
110 FOR X=LEN(A$) TO 1 STEP -1
120 PRINT MID$(A$,X,1);
130 NEXT X
```

The loop steps backwards from the length of the string to 1. The function `MID$(A$,X,1)` identifies the substring in `A$` consisting of one character starting at character number `X`. This isolates a single character.

With an `END` statement added, the complete program is

```
100 INPUT A$
110 FOR X=LEN(A$) TO 1 STEP -1
120 PRINT MID$(A$,X,1);
130 NEXT X
140 END
```

You may wish to run this program using your own name as input.

### Example 2 - Word Count

The number of words in a phrase can be determined from the number of spaces (assuming that the only purpose of a space is to separate words). The following program prints the number of words in the input string.

```

100 INPUT A$
110 LET S=0
120 FOR I=1 TO LEN(A$)
130 IF MID$(A$,I,1) <> " " THEN 150
140 LET S=S+1
150 NEXT I
160 PRINT "WORD COUNT = ";S+1
170 END

```

Study the program until you see exactly how it works. Try out the program by typing in a sentence without commas. Verify that it works correctly.

### Example 3 - Replacement Code

Suppose you want a program to encode a sentence. A simple-minded way to construct a code (which, incidentally, could be broken very rapidly with computers) is to replace each character in the message with another. You can do this most easily by referring to the ASCII character set. However, let's do this one "from scratch."

The first part of the program calls for the input of the string to be coded and sets up the conversion scheme.

```

100 LET B$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ ."
110 LET C$ = "ETAUZHBCW KPSYDFGXIMJLONQU.R"
120 INPUT A$

```

B\$ contains the characters you can use in the input string to be coded. Notice that commas are not allowed. C\$ is the replacement key. An A in the input string is replaced by an E, F is replaced by B, J by a space, and so on.

Now you can program the computer to examine each character and do the replacement.



```

130 FOR I=1 TO LEN(A$)
140 FOR J=1 TO 29
150 IF MID$(A$,I,1) <> MID$(B$,J,1) THEN 180
160 PRINT MID$(C$,J,1);
170 GOTO 190
180 NEXT J
190 NEXT I

```

The outer I loop steps through each character in A\$. The inner J loop compares the Ith character of A\$ to the character in B\$ until a match is found at the Jth character. When this happens, the coded Jth character of C\$ is printed, and the program goes on to the next character in A\$.

You finish the program with

```

200 PRINT
210 END

```

The complete program is

```

100 LET B$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ ."
110 LET C$ = "ETAUZHBCW KPSYDFGXIMJLONQU.R"
120 INPUT A$
130 FOR I=1 TO LEN(A$)
140 FOR J=1 TO 29
150 IF MID$(A$,I,1) <> MID$(B$,J,1) THEN 180
160 PRINT MID$(C$,J,1);
170 GOTO 190
180 NEXT J
190 NEXT I
200 PRINT
210 END

```

You can change the code by rearranging the characters in C\$. To and see how a coded message looks, try running the program.

#### **Example 4 - A Sales Chart as a String Array**

The following is a chart of a retail store's sales of Commodore 64 computers. The sales are given by salesperson for each of four quarters.

		Quarter			
		1	2	3	4
Name	Jack	7	9	11	17
	Fred	5	10	15	20
	Mary	8	7	21	14

This table can be viewed as a string array of three rows and six columns. Your task is to write a program that asks for the name of a salesperson, computes the total unit sales for that person and prints out the result.

First you dimension the array.

```
100 DIM UNITSALES$(3,5)
```

Then you place the data in the program.

```
110 DATA JACK,7,9,11,17
120 DATA FRED,5,10,15,20
130 DATA MARY,8,7,21,14
```

Now enter the data in the string array UNITSALES\$.

```
140 FOR R=1 TO 3
150 FOR C=1 TO 5
160 READ UNITSALES$(R,C)
170 NEXT C
180 NEXT R
```

Now ask for the name of the salesperson.

```
190 PRINT "ENTER NAME OF SALESPERSON ";
200 INPUT N$
```

Next determine the row number of the salesperson.

```
210 FOR R=1 TO 3
220 IF N$=UNITSALES$(R,1) THEN ____ (sum sales)
230 NEXT R
```

If a name is entered erroneously, then you will need the following lines:

```

240 PRINT "NO SUCH SALESPERSON ";N$
250 GOTO 190

```

Now compute the total number of computers sold.

```

300 REM COMPUTE UNIT SALES
310 LET S=0
320 FOR C=2 TO 5
330 LET S=S + VAL(UNITSALES$(R,C))
340 NEXT C
350 PRINT N$;" SOLD";S;" COMPUTERS"
360 END

```

Since the array you are using is a string array, the numbers in the array are strings. Thus, in line 330 you must use the VAL function to convert these strings to numbers. Once you have the table as a string array, there are many types of questions you can use BASIC programming to answer. The table can, of course, be much larger.

The complete program follows.

```

100 DIM UNITSALES$(3,5)
110 DATA JACK,7,9,11,17
120 DATA FRED,5,10,15,20
130 DATA MARY,8,7,21,14
140 FOR R=1 TO 3
150 FOR C=1 TO 5
160 READ UNITSALES$(R,C)
170 NEXT C
180 NEXT R
190 PRINT "ENTER NAME OF SALESPERSON ";
200 INPUT N$
210 FOR R=1 TO 3
220 IF N$=UNITSALES$(R,1) THEN 300
230 NEXT R
240 PRINT "NO SUCH SALESPERSON ";N$
250 GOTO 190
300 REM COMPUTE UNIT SALES
310 LET S=0
320 FOR C=2 TO 5
330 LET S=S + VAL(UNITSALES$(R,C))
340 NEXT C

```

```

350 PRINT N$; " SOLD "; S$; " COMPUTERS"
360 END

```

## 8—5 PROBLEMS

1. Write a program to call for the input of a string, then print the string in a column.
2. Write a program to call for the input of a string, then print the string diagonally so that each character is one line below and one character to the right of the previous character.
3. Write a program to count the number of vowels in an input string.
4. Write a program to call for the input of a string, then print the words in the string in a column.
5. Ask for a sentence to be input. Use this sentence to generate a new string that has all the spaces removed. Then print the new string.
6. Write a program to call for the input of a string consisting of upper case letters. Then convert all the uppercase letters to lowercase and print the string back. You may need to refer to the ASCII character set in the *Commodore 64 User's Guide*, Appendix F.
7. Assume that five sentences are to be entered one at a time. Write a program to count the number of times the word *the* appears in the five sentences.
8. If at the input prompt for the program below you type in the string ABCDEFGH, what will be output?

```

100 INPUT A$
110 FOR J=1 TO LEN(A$) STEP 2
120 PRINT MID$(A$, J, 1);
130 NEXT J
140 END

```

9. Write a program to call for the input of a string and count the number of times the character I is followed by the character N.
10. Count how frequently each of the 26 letters of the alphabet (you may assume that they are all uppercase) occur in 10 sentences typed in at the keyboard. Do not count spaces or punctuation marks. Write a program to compute and print a table consisting of each of the letters and the number of times it occurred in the sentences. Do you think such a table could help you identify an author?

11. Change the program in Example 4 so that it determines the unit sales for each salesperson without asking for the name of the salesperson.
12. Change the program in Example 4 so that it determines the total unit sales in the third quarter.

### 8-6 PRACTICE TEST

1. How are string variables identified in BASIC?

---

2. If A\$ = "KITTY" and B\$ = "KITTYCAT", then A\$ > B\$. True or false?

---

3. If A\$ = "HOW NOW BROWN COW", write a function that will extract NOW BROWN.

---

4. Write a program to call for the input of a string and then print the string again and again, deleting one character each time, until nothing is left. If, for example, you typed in PIECE OF CAKE, the computer should print out

```

PIECE OF CAKE
PIECE OF CAK
PIECE OF CA
PIECE OF C
PIECE OF
PIECE OF
PIECE O
PIECE
PIECE
PIEC
PIE
PI
P

```

5. What will be printed if you run the following program?

```
100 FOR N=65 TO 90
110 FOR M=65 TO N
120 PRINT CHR$(M);
130 NEXT M
140 PRINT
150 NEXT N
160 END
```

---

# **“DO-IT-YOURSELF” FUNCTIONS AND SUBROUTINES**

## **9—1 OBJECTIVES**

In this chapter you will learn to program the computer to perform suboperations. You can do this through either program segments or special one-line instructions.

### **Defining “Do-It-Yourself” Functions**

You have already used functions that are built into Commodore 64 BASIC. You will learn how to use the DEF statement to define your own functions.

### **Understanding Subroutines**

When you need to repeat complicated operations in a program, subroutines are very useful. You will learn how to set up and use subroutines in BASIC programs.

### **Working with Program Examples**

As always you will apply what you learn in the program examples.

## **9—2 DISCOVERY EXERCISES**

1. Turn on the TV set and the computer. Enter the following program:

```
100 DEF FNA(X)=5*X+4
110 LET X=2
120 LET Y=5*X+4
130 PRINT Y,FNA(2)
140 END
```

Run the program and record the output below.

---

2. Edit line 130 as follows:

```
130 PRINT Y,FNA(X)
```

Display the program in memory. What do you think will happen if we run this program?

---

Run the program. What happened?

---

3. Type

```
110 LET X=5
```

Display the program and study it. Now what will be the output if you run the program?

---

See if you were right. Run the program and record what happened.

---

4. Edit line 130 as follows:

```
130 PRINT Y,FNA(5)
```

Display the program. What do you think the program will do now?

---



Run the program and write down the output.

---

5. Notice that the expressions after the equal signs in lines 100 and 120 of your program are the same. In one version of the program, you printed Y and FNA(X) and saw that they were the same. Now follow up on this information. Clear the program in memory. Then enter the following program:

```
100 DEF FNA(X)=X↑2
110 DEF FNB(X)=3*X
120 DEF FNC(X)=X+2
130 LET X=1
140 PRINT FNA(X);FNB(X);FNC(X)
150 END
```

Study the program carefully. What do you think the computer will print if you run the program?

---

Now run the program and write down what happened.

---

Do not change your program, but think about what would happen if you substituted 1 for X in the expressions on the right side of lines 100, 110, and 120. Write down the numbers you would obtain.

---

6. Type

```
130 LET X=2
```

Display the program. What will happen if you run the program now?

---

See if you were right. Run the program and record the results below.

---

7. Type

•     130 LET X=3

Now what will happen if you run the program?

---

Verify your answer by running the program and recording what happened.

---

8. Now on to some more ideas you can explore with this program. Type

```
130 LET X=1
140 PRINT FNC(X+4);FNB(2);FNA(X)
```

Display the program. Write down what you think will be printed if the program is executed.

---

Run the program and record the output.

---

9. Try a slightly different variation on the theme you have been exploring. Type

```
140 PRINT FNA(X);FNB(FNA(X))
```

Display the program and study it carefully. Try to figure out what will be printed out when the program is executed. Record your answer below.

---

Run the program and see if you were right. In the space below, write what happened.

---

10. One last point on this matter. Type

```
130 LET X=4
140 PRINT FNA(X);FNC(X);FNA(SQR(X))
```

Now what will happen in the program?

---

Run the program and record what happened?

---

11. Clear the program in memory. Enter the following program:

```
100 PRINT "A";
110 GOSUB 200
120 PRINT "B";
130 GOSUB 300
140 PRINT "C";
150 END
200 PRINT 1;
210 RETURN
300 PRINT 2;
310 RETURN
```

This program has two statements you haven't seen so far, GOSUB and RETURN. The only purpose of the program is to give you practice in tracing these new statements. Run the program and record the output.

---

Compare the output with the program lines that generate it.

12. To which statement does the GOSUB statement in line 110 transfer the program? (Hint: Look at the output in step 11.)

---

13. To which statement does the RETURN statement in line 210 transfer the program? (Hint: Again, examine the output in step 11.)

---

14. The line numbers below trace the flow of the program as it is executed.

Line Number	What Happens
100	Prints A
110	Transfers to line 200
200	Prints 1
210	Transfers to line 120
120	Prints B
130	Transfers to line 300
300	Prints 2
310	Transfers to line 140
140	Prints C
150	Ends program

Study this sequence carefully and follow it through the program. Do you see the purpose of the GOSUB and RETURN statements? What about the placement of the END statement?

---

15. Clear the program in memory. Enter the following program:

```

100 REM PROGRAM TO DEMONSTRATE SUBROUTINES
110 DIM X(4)
120 FOR I=1 TO 4
130 READ X(I)
140 NEXT I
150 REM SORT
160 GOSUB 300
170 REM PRINT OUT SORTED ARRAY
180 FOR I=1 TO 4
190 PRINT X(I);
200 NEXT I
210 PRINT
220 LET X(3)=7
230 REM SORT AGAIN
240 GOSUB 300
250 REM PRINT OUT SORTED ARRAY
260 FOR I=1 TO 4
270 PRINT X(I);
280 NEXT I
290 END
295 DATA 2,1,5,6
300 REM SUBROUTINE TO SORT
310 FOR I=1 TO 3
320 IF X(I+1) > X(I) THEN 370
330 LET TEMP=X(I+1)
340 LET X(I+1)=X(I)
350 LET X(I)=TEMP
360 GOTO 310
370 NEXT I
380 RETURN
    
```

Display the program and check that you have entered it correctly. This program shows how a subroutine might be used. The subroutine in lines 300 through 380 sorts the array X into ascending order. Run the program and record the output.

---

The original array is

2 1 5 6

You can see this by checking the DATA statement in line 295. In line 160, the program jumps to the subroutine, which sorts the numbers. When the program returns to line 170, the sorted array is

1 2 5 6

In line 220, you change the third element of the array, then branch to the subroutine for another sorting. After the return to line 250, the sorted array

1 2 6 7

is printed out. Finally, the END command in line 290 stops the running of the program. Clearly, you could sort the array X as often as you wished merely by inserting a GOSUB 300 statement. This is certainly more efficient than repeating the sorting instructions each time you want the array sorted.

16. Now look at another statement. Clear the memory and type the following program.

```

100 PRINT "ENTER A NUMBER BETWEEN 1 AND 5."
110 PRINT "ENTER 5 TO QUIT."
120 INPUT N
130 ON N GOSUB 1000,2000,3000,4000
140 IF N=5 THEN END
150 PRINT
160 GOTO 100
1000 PRINT "ONE"
1010 RETURN
2000 PRINT "TWO"
2010 RETURN
3000 PRINT "THREE"
3010 RETURN
4000 PRINT "FOUR"
4010 RETURN

```

What word will be printed if you enter 3?

---

Run the program. Were you right?

---

17. This completes the computer work for this chapter. Turn off the computer and the TV set.

### 9-3 DISCUSSION

#### Defining "Do-It-Yourself" Functions

The DEF (for "define") statement permits user-specified functions in BASIC programs in addition to those functions (SQR, INT, etc.) already built into the language. The form of all DEF statements is the same.

```

<line#> DEF FN<letter><variable> = <expression using variable>
100 DEF FNP(X)=X↑2 - 3*X
    
```

Since any valid variable name could follow FN, you could conceivably define many functions in a single program.

If you use the sample DEF statement in line 100 in a program, and later used the expression FNP(2), the computer would "look up" the definition of FNP in line 100 and then substitute 2 for X on the right side of the equal sign in the DEF statement, with this result

$$FNP(2) = -2$$

Likewise, if MINUTES = 5, then

$$FNP(MINUTES) = 10$$

You can use the built-in functions of BASIC in DEF statements. For example,

```
100 DEF FNB(Y)=SQR(Y↑1.5) + 3*Y
```

is OK. Furthermore, you may use other defined functions in a DEF statement. For example,

```
100 DEF FNB(Y)=FNA(Y) + SQR(Y)
```

is correct provided you have defined function FNA previously. The primary purpose of user-specified functions is to simplify programming by avoiding repeated use of complicated expressions. The wise programmer is alert for opportunities to save effort with DEF statements.

- Use the DEF statement to define your own functions.

### Understanding Subroutines

One limitation of a DEF statement is that it can occupy no more than a single line. You are bound to encounter more complicated situations that require the repetition of a multiline process. Subroutines are very useful in such an event. The diagram below shows how you might use a subroutine in a program.

Main program begins	_____
	_____
	_____
	200 GOSUB 1000
	210
	_____
	_____
	_____
	350 GOSUB 1000
	360
	_____
Main program ends	430 END
Subroutine begins	1000 REM SUBROUTINE
	_____
	_____
	_____
End of subroutine	1150 RETURN

When the computer reaches the GOSUB in line 200, it jumps to line 1000 and executes the subroutine that begins on that line.



When the computer encounters the RETURN in line 1150, it returns to line 210—the next higher line number after the GOSUB that put it in the subroutine. Then the computer proceeds through the main program to the GOSUB in line 350, which again causes it to branch to the subroutine in line 1000. This time, the RETURN statement sends the computer back to line 360 in the program.

Of course, you can use GOSUB 1000 as many times as you want or you can use as many subroutines as you need. Generally, the top part of the program is the main program, and the subroutines are grouped together at the end. There is a good reason for this. You want the computer to perform the subroutines only when they are called by a GOSUB. Thus, after the main program is finished, you put an END statement in the program.

### ■ Transfer to subroutines with GOSUB.

It is possible, and sometimes desirable, to jump to a subroutine from a subroutine. The diagram below indicates how the computer treats such an event.

Main program

\_\_\_\_\_  
\_\_\_\_\_

Subroutine 1

400 GOSUB 800 → 800

410

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

550 END

Subroutine 1

820 GOSUB 900 → 900

830

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

880 RETURN

Subroutine 2

820 GOSUB 900 → 900

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

990 RETURN

Note that control passes from 400 to 800, down to 820, to 900, and then down to the RETURN in line 990. Of course, the question is, does the RETURN in line 990 return control to line

410 or to line 830? The rule is that the computer returns to the next statement after the GOSUB that put it in the subroutine. The computer is in subroutine 2 because of the GOSUB in line 820; therefore the RETURN in line 990 branches back to line 830. The same rule applies when the computer reaches the RETURN in line 880. At that point the computer is in subroutine 1 and arrived there from the GOSUB in line 400. Thus, the RETURN in line 880 branches back to line 410.

■ **Use RETURN to branch back from subroutines.**

The ON...GOSUB statement allows you to send the computer to one of numerous subroutines depending on the value of a numeric variable. For example, the statement

```
140 ON N GOSUB 300,400,500
```

will go to the subroutine at 300 if N is 1, to the subroutine at 400 if N is 2, or to the subroutine at 500 if N is 3.

At this point it may not be clear to you why subroutines are valuable. As you acquire more skill as a programmer, the need for subroutines will become more evident. Subroutines are one of the most useful tools available to the programmer.

## 9-4 WORKING WITH PROGRAM EXAMPLES

### Example 1 - Rounding Off Dollar Values to Cents

Business applications generally involve calculations in dollars and cents. Since the computer normally handles seven significant figures, it might print a dollar amount such as 23.15793. Ordinarily, such amounts are rounded off to the nearest cent, or 23.16.

This is an ideal application for a user-defined function. You can write a program that will produce the following output when executed:

```
LABEL PRICE? (You type in price)
10% DISCOUNT IS (Computer prints discount price)
15% DISCOUNT IS (Computer prints discount price)
20% DISCOUNT IS (Computer prints discount price)
```

All dollar values typed out should be rounded off to the nearest cent.

First you define a function to do the rounding off. Such a function is

```
100 DEF FNR(X)=INT(X*100+.5)/100
```

To see how this function works, suppose  $X = 23.1597$ . Follow this value through the expression to see what happens.

$X*100$	=	2315.97
$X*100+0.5$	=	2316.47
$\text{INT}(X*100+0.5)$	=	2316
$\text{INT}(X*100+0.5)/100$	=	23.16

Therefore, 23.1597 is correctly rounded up to 23.16.

Now suppose that  $X = 23.1547$ . Then

$X*100$	=	2315.47
$X*100+0.5$	=	2315.97
$\text{INT}(X*100+0.5)$	=	2315
$\text{INT}(X*100+0.5)/100$	=	23.15

with the result that 23.1547 is correctly rounded down to 23.15.

The next few lines of the program are self-explanatory.

```
110 PRINT "LABEL PRICE";
120 INPUT Z
130 PRINT "10% DISCOUNT IS";FNR(.90*Z)
140 PRINT "15% DISCOUNT IS";FNR(.85*Z)
150 PRINT "20% DISCOUNT IS";FNR(.80*Z)
```

You can loop back to the beginning with

```
160 GOTO 110
```

and then end the program.

```
170 END
```

The complete program is

```
100 DEF FNR(X)=INT(X*100+.5)/100
110 PRINT "LABEL PRICE";
120 INPUT Z
130 PRINT "10% DISCOUNT IS";FNR(.90*Z)
140 PRINT "15% DISCOUNT IS";FNR(.85*Z)
150 PRINT "20% DISCOUNT IS";FNR(.80*Z)
160 GOTO 110
170 END
```

Lines 130, 140, and 150 use the defined function. For a 10 percent discount, the selling price is 90 percent of the original price  $Z$ . Hence, the computer prints  $FNR(.90*Z)$ , which rounds off the value to the nearest cent. Note the economy of using the defined function rather than writing out the expression in line 100 each time you want to print a rounded dollar amount.

### Example 2 - Carpet Estimating

Now write a program that uses a subroutine to compute the price of installed carpet. Suppose that there are four grades of carpet and that each is discounted as the quantity of carpet ordered increases. Assume that the price structure is as follows:

	1	2	3
<b>Grade</b> A	\$10.00	\$ 8.50	\$ 7.25
B	13.25	12.00	9.75
C	16.00	14.00	11.25
D	20.00	17.20	15.25

The column headings mean the following:

1. First 15 square yards
2. Any part of the order exceeding 15 but not more than 25 square yards
3. Anything over 25 square yards

When executed, the program should produce the following output:

```
HOW MANY ROOMS? (You type in number of rooms)
FOR EACH ROOM, TYPE IN LENGTH
AND WIDTH IN FEET,
SEPARATED BY A COMMA
```

```
ROOM                DIMENSIONS
```

```
1                (You type in dimensions)
2                (You type in dimensions)
```

(Loop until all rooms are entered)

```
(Computer types out number) SQUARE YARDS REQUIRED
CARPET GRADE                COST OF ORDER
A                (Computer types out,etc.)
B
C
D
```

Before proceeding further with the program, think a bit about the output. Since the output is in dollars and cents, you may as well use the DEF function from example 1 to round off the answers properly. Begin the program with that defined function.

```
100 DEF FNR(X)=INT(X*100+.5)/100
```

The next few lines follow without difficulty.

```
110 PRINT "HOW MANY ROOMS";
120 INPUT N
130 PRINT "FOR EACH ROOM, TYPE IN LENGTH"
140 PRINT "AND WIDTH IN FEET,"
150 PRINT "SEPARATED BY A COMMA"
160 PRINT
170 PRINT "ROOM", "DIMENSIONS"
180 PRINT
```

Now you can call for the input of the room dimensions. Use the variable A to keep track of the area of the rooms. Remember that the area of a room is its length times its width.

```
190 LET A=0
200 FOR I=1 TO N
210 PRINT I,
220 INPUT L,W
230 LET A=A+L*W
240 NEXT I
```

Since the total room area is now in square feet, divide the square feet by 9 to convert to square yards, and then have the computer print the number of square yards of carpeting required.

```
250 LET A=A/9
255 PRINT
260 PRINT A;"SQARE YARDS REQUIRED"
```

At this point you may as well use DATA statements to include the price table in the program.

```
270 DATA 10,8.5,7.25
280 DATA 13.25,12,9.75
290 DATA 16,14,11.25
300 DATA 20,17.2,15.25
```

Next print out the headings required for the price printout.

```
310 PRINT
320 PRINT "CARPET GRADE","COST OF ORDER"
330 PRINT
```

You are now at the point in the program where the subroutine will be useful. Since you don't know precisely where the subroutine should begin, simply use a large line number and correct it later if you need to.

```
340 REM COMPUTE PRICE FOR GRADE A
350 GOSUB 800
```

Write the subroutine now. First, you need the prices of each of the three grades of carpet. Program the computer to read them from the DATA statements.

```
800 REM SUBROUTINE TO COMPUTE CARPET PRICE
810 READ C1,C2,C3
```

Next write program lines that test whether the carpet area is less than 15, between 15 and 25, or more than 25 square yards and then compute the price accordingly.

```
820 IF A>25 THEN 860
830 IF A>15 THEN 880
840 LET P=C1*A
850 GOTO 890
860 LET P=15*C1 + 10*C2 + (A-25)*C3
870 GOTO 890
880 LET P=15*C1 + (A-15)*C2
890 RETURN
```

Study this program segment to convince yourself that the price is computed correctly. Then return to the main program and print out the first price.

```
360 PRINT "A";TAB(25);FNR(P)
```

Once this pattern is established, the rest of the main program follows easily.

```
370 REM COMPUTE PRICE FOR GRADE B
380 GOSUB 800
390 PRINT "B";TAB(25);FNR(P)
400 REM COMPUTE PRICE FOR GRADE C
410 GOSUB 800
420 PRINT "C";TAB(25);FNR(P)
430 REM COMPUTE PRICE FOR GRADE D
440 GOSUB 800
450 PRINT "D";TAB(25);FNR(P)
460 END
```

You need the END statement in line 460 to prevent the program from falling into the subroutine. The value of the subroutine should be clear. Without subroutines, the program would require many more lines. You would need to replace each of the four GOSUB statements with as many statements as are in the subroutine.

The complete program is

```
100 DEF FNR(X)=INT(X*100+.5)/100
110 PRINT "HOW MANY ROOMS";
120 INPUT N
130 PRINT "FOR EACH ROOM, TYPE IN LENGTH"
140 PRINT "AND WIDTH IN FEET,"
150 PRINT "SEPARATED BY A COMMA"
160 PRINT
170 PRINT "ROOM","DIMENSIONS"
180 PRINT
190 LET A=0
200 FOR I=1 TO N
210 PRINT I,
220 INPUT L,W
230 LET A=A+L*W
240 NEXT I
250 LET A=A/9
255 PRINT
260 PRINT A;"SQUARE YARDS REQUIRED"
270 DATA 10,8.5,7.25
280 DATA 13.25,12,9.75
290 DATA 16,14,11.25
300 DATA 20,17.2,15.25
310 PRINT
320 PRINT "CARPET GRADE","COST OF ORDER"
330 PRINT
340 REM COMPUTE PRICE FOR GRADE A
350 GOSUB 800
360 PRINT "A";TAB(25);FNR(P)
370 REM COMPUTE PRICE FOR GRADE B
380 GOSUB 800
390 PRINT "B";TAB(25);FNR(P)
400 REM COMPUTE PRICE FOR GRADE C
410 GOSUB 800
420 PRINT "C";TAB(25);FNR(P)
430 REM COMPUTE PRICE FOR GRADE D
440 GOSUB 800
450 PRINT "D";TAB(25);FNR(P)
460 END
800 REM SUBROUTINE TO COMPUTE CARPET PRICE
810 READ C1,C2,C3
```



```

820 IF A>25 THEN 860
830 IF A>15 THEN 880
840 LET P=C1*A
850 GOTO 890
860 LET P=15*C1 + 10*C2 + (A-25)*C3
870 GOTO 890
880 LET P=15*C1 + (A-15)*C2
890 RETURN

```

### 9-5 PROBLEMS

1. Study the program below and write down what it will print if you run the program.

```

100 DEF FNA(X)=2+X
110 DEF FNB(Y)=10*Y
120 DEF FNC(Z)=Z↑2
130 LET R=2
140 LET S=3
150 LET T=5
160 PRINT FNC(T);FNA(S);FNB(R)
170 LET R=S+T
180 PRINT FNA(R) + FNB(S) + FNC(T)
190 END

```

2. What will be the output if you run the program below?

```

100 DEF FNX(A)=6*A
110 DEF FNY(B)=B+10
120 DEF FNZ(C)=C↑3
130 READ P,Q,R
140 DATA 1,2,3
150 PRINT FNX(R);FNZ(P);FNY(Q)
160 PRINT FNY(P+Q) + FNX(R)
170 END

```

3. The area of a circle is pi times R squared, and the volume of a sphere is  $\frac{4}{3}$  times pi times R cubed. Pi is 3.14159, and R is either the radius of the circle or the radius of the sphere. Define two DEF statements, one for the circle and one for the sphere. Set up a FOR NEXT loop for R from 1 to 10 in steps of 0.5. Use the defined

functions to print out a table of areas and volumes for each of the values of R.

4. What will be the output if you run the following program?

```
100 DIM A(5)
110 FOR I=1 TO 5
120 READ A(I)
130 NEXT I
140 DATA 6,2,7,1,3
150 GOSUB 500
160 LET A(3)=10
170 GOSUB 500
180 LET A(5)=8
190 GOSUB 500
200 END
500 FOR I=1 TO 4
510 LET A(I)=A(I+1)
520 NEXT I
530 GOSUB 600
540 RETURN
600 FOR J=1 TO 5
610 PRINT A(J);
620 NEXT J
630 RETURN
```

5. What will be the output if you run the program below?

```
100 LET X=10
110 GOSUB 500
120 PRINT S
130 LET X=X/2
140 GOSUB 500
150 PRINT S
160 LET X=X+3
170 GOSUB 500
180 PRINT S
190 END
500 LET S=0
510 FOR Y=1 TO X
520 LET S=S + Y
530 NEXT Y
540 RETURN
```

6. Assume that a one-dimensional array Z contains numbers to be added together. The first element of the array Z(1) gives the number of elements that follow in the array and are to be summed. Write a subroutine beginning in line 800 to compute the sum of the elements after Z(1). Assign the sum to the variable T. Terminate the subroutine with a RETURN statement. Assume that the array Z has been properly dimensioned and that the values in the array have been loaded in the main program.
7. X is a one-dimensional array. The first element of the array X(1) gives the number of pieces of data that follow in the array. Write a subroutine beginning in line 500 to search that array for the largest value. Assign this value to the variable L. Terminate the subroutine with a RETURN statement. Assume that the array X has been properly dimensioned and loaded with numbers elsewhere.
8. Suppose that as part of a printout you need a series of 40 asterisks (\*) in a straight line across the page. Write a subroutine beginning in line 1000 to do this. Terminate the subroutine with a RETURN statement.
9. Assume that a one-dimensional array Y is loaded with numbers. The first element Y(1) gives the number of elements to follow. Write a subroutine to calculate the mean (M) and standard deviation (S) of the numbers in the array that follow the first element. Begin the subroutine in line 900 and terminate with a RETURN statement. The formulas for calculation of the mean and standard deviation are given below.

$$\text{Mean} = (\text{Sum of values})/N$$

$$\text{Standard deviation} = \sqrt{\frac{N \times (\text{sum of squares of values}) - (\text{sum of values})^2}{N \times (N - 1)}}$$

## 9-6 PRACTICE TEST

1. If  $\text{DEF FNA}(X) = \text{SQR}(X) + 3 * X$ ,  $Z = 2.5$ , and  $W = 10$ , what is
  - a.  $\text{FNA}(1)$

---

b. FNA(4)

---

c. FNA(9)

---

d. FNA(Z\*W)

---

2. What will be the output if you run the following program?

```
100 DEF FNR(X)=X*X
110 DEF FNS(X)=3*X
120 DEF FNT(Y)=Y+1
130 LET A=1
140 PRINT FNT(A);FNR(A);FNS(A)
150 LET M=4
160 PRINT FNR(SQR(M))
170 END
```

3. With regard to subroutines:

a. How do you pass control from the main program to the subroutine?

---

b. How do you pass control from the subroutine back to the main program?

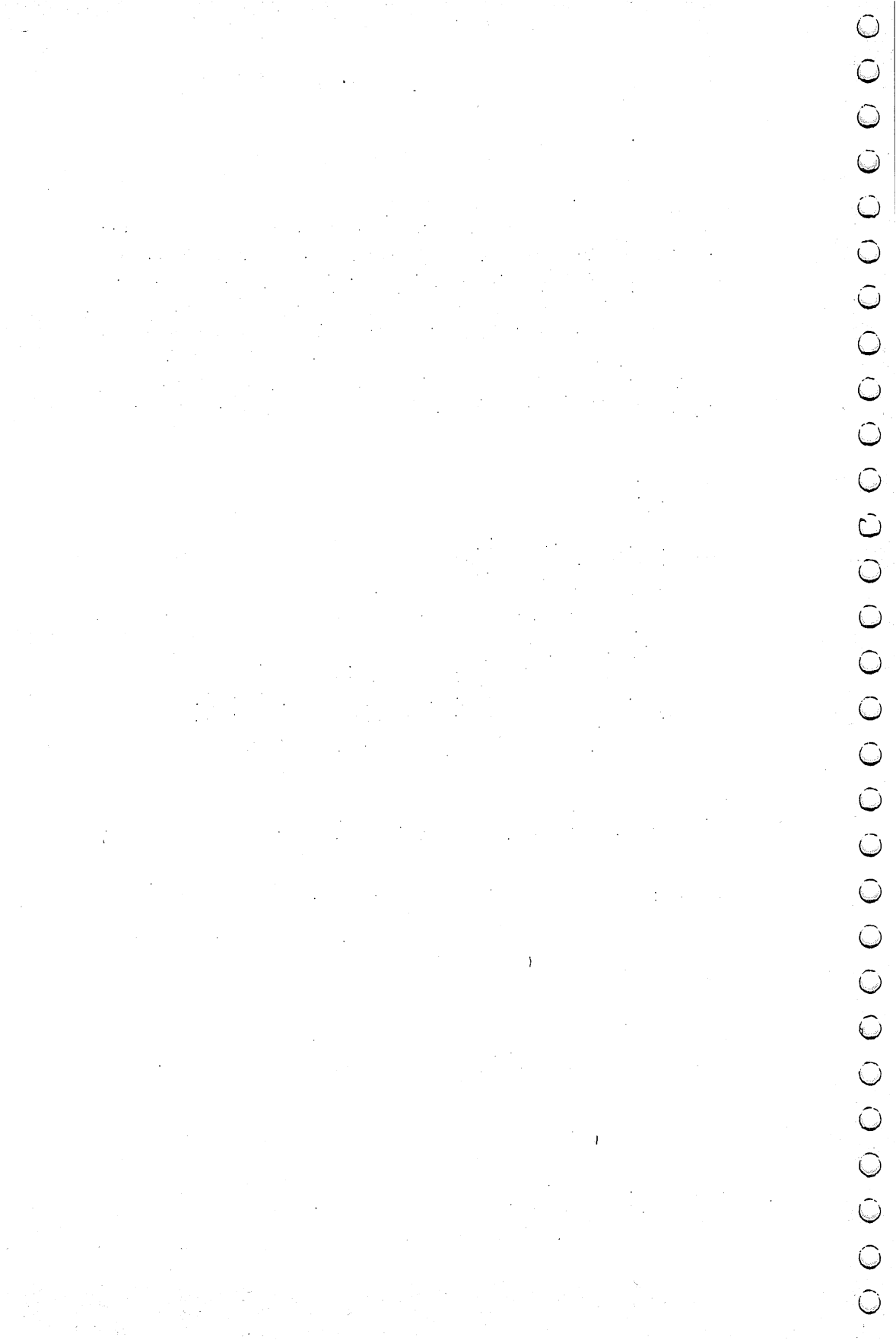
---

4. What will be printed if you run the following program?

```

100 LET A=1
110 GOSUB 200
120 LET A=A+4
130 GOSUB 200
140 LET A=A-2
150 GOSUB 200
160 END
200 REM SUBROUTINE
210 IF A<2 THEN 250
220 IF A=3 THEN 270
230 PRINT "RED"
240 GOTO 280
250 PRINT "WHITE"
260 GOTO 280
270 PRINT "BLUE"
280 RETURN
    
```

---



# CHARACTER GRAPHICS AND COLORS

## 10—1 OBJECTIVES

### **Controlling Programs**

Graphics programming is easier if more elaborate program control is possible. You will learn additional techniques for program control.

### **Placing Characters**

Graphics programming requires that you be able to place a character at any position on the screen. You will learn to use a subroutine to do this.

### **Working with the Graphics Features of the Commodore 64**

The Commodore 64 has many interesting graphics characters and display modes. You will learn to use the reverse character feature and some of the graphics characters.

### **Controlling Color**

Using color in graphics screen displays improves appeal and impact. You will learn to control the color of any character printed on the screen.

### **Working with Program Examples**

You will use what you learn to explore the Commodore 64 graphics and color features.

**10-2 DISCOVERY EXERCISES**

1. Turn on the TV set and the computer. Type in the following program.

```
100 INPUT A
200 PRINT "A",A=1,A=2,A=3
300 GOTO 100
400 END
```

2. Run the program. At the input prompt, type 2. Is the first, second, or third number a -1?

---

For the number 2 you input for A, which of the statements A=1, A=2, and A=3 is true.

---

If 3 is entered at the input, what list of numbers will be displayed?

---

Input 3 to see if you were correct.

3. Interrupt the program. Add the following lines to the program.

```
110 PRINT TAB(10);1,2,3
120 PRINT
210 PRINT
220 PRINT "B",B=1,B=2,B=3
250 PRINT
```



4. Now change line 100 as follows.

```
100 INPUT A,B
```

Display the program. The program should look like this.

```
100 INPUT A,B
110 PRINT TAB(10);1,2,3
120 PRINT
200 PRINT "A",A=1,A=2,A=3
210 PRINT
220 PRINT "B",B=1,B=2,B=3
250 PRINT
300 GOTO 100
400 END
```

Run the program. At the input prompts, type

```
1,1
2,3
3,3
```

For which pairs of numbers are the -1's in the same column?

---

5. Interrupt the program. Add lines 230 and 240 as follows.

```
230 IF A=1 AND B=1 THEN PRINT
    "CONDITION IS SATISFIED":PRINT:GOTO 100
240 PRINT "CONDITION IS NOT SATISFIED"
```

Display the program. Notice the colons before and after the second PRINT and GOTO 100 at the the end of line 230. Here two PRINT statements and a GOTO statement will be performed if the condition is true. If the program is run, which pair of numbers must be input so that "CONDITION IS SATISFIED" will be printed out?

---

Run the program to see if you were correct.

6. Interrupt the program. Finally, change line 230 as follows.

```
230 IF A=1 OR B=1 THEN PRINT  
"CONDITION IS SATISFIED":PRINT:GOTO 100
```

Display the program. What pairs of numbers will cause "CONDITION IS SATISFIED" to be typed out?

---

Try your pairs of numbers to see if you were correct. You should have written down five pairs of numbers. Is the condition satisfied if both **A** and **B** are input as 1?

- 
7. On to another topic. Clear the memory and type in the following program.

```
100 GET A$  
110 IF A$="" THEN 100  
120 PRINT ASC(A$)  
130 IF A$<>"Q" THEN 100  
140 END
```

Fill in the following table by pressing the indicated keys/key combinations.

Key Combination	Function	ASCII code
SHIFT CLR/HOME	Clear screen	_____
CLR/HOME	Place cursor in the HOME position in the upper left corner of the screen	_____
SHIFT ⇐CRSR⇒	Cursor left	_____
⇐CRSR⇒	Cursor right	_____
SHIFT ↑CRSR↓	Cursor up	_____
↑CRSR↓	Cursor down	_____

8. Fill in the following table by holding down the CTRL key and pressing the indicated number.

Key Combination	Character mode	ASCII code
CTRL 1	Black	_____
CTRL 2	White	_____
CTRL 3	Red	_____
CTRL 4	Cyan	_____
CTRL 5	Purple	_____
CTRL 6	Green	_____
CTRL 7	Blue	_____
CTRL 8	Yellow	_____
CTRL 9	Reverse On	_____
CTRL 0	Reverse Off	_____

9. Interrupt the program. In direct mode, type

```
PRINT "
```

Now press each of the keys indicated in steps 7 and 8. Notice that characters are now printed out. Now press **RETURN**. What happened?

---

Where the CHR\$ function and the ASCII code are used, you may, if you wish, use "(the desired key combination)". For example, type

```
PRINT "(SHIFT|CLR/HOME)"
```

What happened?

---

Press **RUN/STOP/RESTORE** to return to the standard character display. Now type to return to the standard

```
PRINT CHR$(147)
```

Recall from the table you filled in in step 7 that the ASCII code for SHIFT|CLR/HOME is 147. Did the screen clear this time also?

---

10. Now on to graphics programming. Clear the memory. Recall that **SHIFT|↑CRSR↓** has ASCII code 17, **←CRSR→** has ASCII code 29, and **CLR/HOME** has ASCII code 19. Type in the following subroutine.

```
1000 REM CURSOR PLACEMENT SUBROUTINE
1010 IF R<1 OR R>25 OR C<1 OR C>40
THEN END
1020 IF R=25 AND C=40 THEN END
1030 PRINT CHR$(19);
1040 IF R=1 THEN 1080
1050 FOR J=1 TO R-1
1060 PRINT CHR$(17);
1070 NEXT J
```

```

1080 IF C=1 THEN 1120
1090 FOR K=1 TO C-1
1100 PRINT CHR$(29);
1110 NEXT K
1120 RETURN

```

Check the subroutine carefully, especially the semicolons at the ends of lines 1030, 1060, and 1100. This subroutine uses the values of **R** and **C** to position the cursor so that the next character will be placed in the **R**th row and **C**th column. ASCII codes of the cursor movement keys are used to move the cursor to the desired position. Lines 1010 and 1020 are used to make sure that the cursor is not moved off the screen.

11. Now type in a program that uses this subroutine.

```

100 PRINT CHR$(147)
110 PRINT CHR$(19);TAB(10);"ENTER
ROW AND COLUMN";
120 INPUT R,C
130 GOSUB 1000
140 PRINT "X";
150 GOTO 110
160 END

```

Display the program. Run the program. At the input prompts, enter the following pairs of numbers.

```

1,1
5,7
5,8
5,40
25,5
25,39
25,40

```

How many characters wide is the screen?

---

How many characters deep is the screen?

---

Using this subroutine, can you place an X in the 25th row and 40th column?

---

12. Now let's change the printed character from an X to a Commodore 64 graphics character. Edit line 140 as follows.

```
140 PRINT CHR$(122)
```

Run the program. Enter the following pairs of numbers at the input prompts.

```
13,20  
20,5
```

Draw the new symbol printed by the program.

---

13. Interrupt the program. Type line 140 as follows.

```
140 PRINT "(SHIFT|Z)"
```

Notice that the diamond is on the right face of the Z key. Run the program. Enter 13,20 at the input prompt. Was the same symbol printed out?

---

14. Now you will look at the reverse character feature of the Commodore 64. Change line 140 as follows.

```
140 PRINT CHR$(18);CHR$(122);
```

Run the program entering 13,20 at input prompt. How has the diamond changed?

---

15. Finally change line 140 as follows.

```
140 PRINT CHR$(18);" ";
```

Here you will print a reverse space. Run the program entering 13,20 at the input prompt. What does a reverse space look like?

---

16. If you do not have a color TV set, go to step 21.
17. Clear the memory of the program with its subroutine. Type in the following program.

```
100 REM ASCII CODES FOR COLOR
110 DATA BLACK,144,WHITE,5,RED,28
120 DATA CYAN,159,PURPLE,156,GREEN,30
130 DATA BLUE,31,YELLOW,158
140 FOR K=1 TO 8
150 READ COLR$,COLR
160 PRINT CHR$(COLR);
170 FOR I=1 TO 80
180 PRINT CHR$(18);" ";
190 NEXT I
200 NEXT K
210 END
```

Display the program. Run the program. You should see bands of colors on the screen in the order given in lines 110, 120, and 130. You may wish to adjust the screen so that the colors more nearly match their names. Printing CHR\$(ASCII code for the color) causes characters printed on the screen to be that color. Notice that READY is very hard to read as it is printed in yellow on a light blue background. To make the screen more readable, press RUN/STOP|RESTORE to return the computer to the start-up screen mode of white on a light blue background.

18. Clear the memory and type in the following program.

```

100 REM ASCII CODES FOR COLORS
110 DATA YELLOW,158, GREEN,30, RED,28
120 FOR K=1 TO 3
130 READ C$(K),C(K)
140 NEXT K
150 PRINT "ENTER COLOR";
160 INPUT COLR$
170 FOR K=1 TO 3
180 IF ASC(COLR$)=ASC(C$(K)) THEN 200
190 NEXT K
200 PRINT "ENTER NUMBER OF SQUARES";
210 INPUT N
220 PRINT CHR$(C(K));
230 FOR I=1 TO N
240 PRINT CHR$(18);" ";CHR$(146);
250 NEXT I
260 PRINT CHR$(154);
270 PRINT
280 GOTO 150
290 END

```

Study the program. In line 260, CHR\$(154) returns the character printing color to the start-up color. In line 180, how many characters of the word you enter are considered by the ASC function (see step 20 in Chapter 8)?

---

Run the program. At the input prompts enter

```

YELLOW,5
GREEN,14
RED,51

```

What happened?

---



19. Now enter

**BLACK, 25**

What color was the horizontal bar this time?

---

20. This concludes the Discovery Exercises. Turn off the computer and the TV set.

### 10-3 DISCUSSION

#### Controlling Programs

Logical (true/false) expressions such as  $A=5$  are evaluated by Commodore 64 BASIC as -1 or 0. The value of  $A=5$  depends on whether the statement "A is equal to 5" is true or false. When the numeric variable **A** is 5, the statement is true and the value of  $A=5$  is -1. When **A** is not 5, the statement is false and the value of  $A=5$  is 0. IF THEN conditions contain such logical expressions.

In the chapter on decisions and branching, you learned about IF THEN statements with simple conditions. In this chapter, you used conditions containing two simple logical expressions separated by AND or OR. For example, the condition in

```
100 IF A=5 OR B=6 THEN PRINT "OK"
```

is satisfied if the value of **A** is 5 or the value of **B** is 6. The condition is also satisfied if both the value of **A** is 5 and the value of **B** is 6.

If AND is used in line 100 you have

```
100 IF A=5 AND B=6 THEN PRINT "OK"
```

Here the condition is satisfied and OK is printed only when the value of **A** is 5 while, at the same time, the value of **B** is 6.

More complicated conditions can be created. For example, in

```
20 IF (R=5 OR C=5) AND D>7 THEN 50
```

the condition is satisfied when either **R** or **C** is 5 while **D** is greater than 7.

You evaluate an arithmetic expression such as  $(A+B)*C$  which contains the arithmetic operations  $+$ ,  $-$ ,  $*$ ,  $/$  using the rules discussed in the chapter on computer arithmetic. You evaluate a logical expression that contains the logical words or operations AND and OR using their standard logical meanings. Depending on the values of the variables in an arithmetic expression, its value can be any of a host of numbers. On the other hand, the values of a logical expression can only be  $-1$  (true) or  $0$  (false).

The colon (:) can be used to separate multiple statements on a single line. You have used this capability only after THEN in IF THEN statements. For example, the program lines

```
100 IF A=5 THEN PRINT "A=5 IS TRUE":
    GOTO 300
200 PRINT "A=5 IS FALSE"
```

will print A=5 IS TRUE and then transfer to line 300 if the condition is satisfied. If the condition is not satisfied, A=5 IS FALSE will be printed. Excessive use of this feature can make programs hard to understand and to troubleshoot.

### Placing Characters

You used the following subroutine to assist you in placing characters on the screen.

```
1000 REM CURSOR PLACEMENT
1010 IF R<1 OR R>25 OR C<1 OR C>40 THEN END
1020 IF R=25 AND C=40 THEN END
1030 PRINT CHR$(19);
1040 IF R=1 THEN 1080
1050 FOR J=1 TO R-1
1060 PRINT CHR$(17);
1070 NEXT J
1080 IF C=1 THEN 1120
1090 FOR K=1 TO C-1
1100 PRINT CHR$(29);
1110 NEXT K
1120 RETURN
```

This subroutine uses the values of **R** and **C** as the row and column position in which you want to place the character. Line 1010 checks

to make sure that the row and column position is on the screen. Line 1020 makes sure that the last row, last column position is not used. If a character is printed in the last screen position, Commodore 64 BASIC issues a RETURN character, which moves all the displayed characters up one line. Line 1030 used the ASCII code CHR\$(19), for CLR/HOME to move the cursor to the HOME position in the upper left corner of the screen. Lines 1050-1070 and lines 1090-1110 print the desired number of right cursor, CHR\$(17), and down cursor, CHR\$(29), to place the cursor in the Rth row and Cth column. Lines 1040 and 1080 take care of the special cases when R is 1 or C is 1. In those cases, no cursor right or cursor down characters should be printed to the screen as the cursor always starts in the HOME position in the first row and first column.

You can change the END statements in 1010 and 1020 depending on the particular results you desire. You might wish to develop subroutines to handle these special cases.

### Working with the Graphics Features of the Commodore 64

Characters can be printed as white on a blue background; or the reverse, blue characters on a white background. This reverse character graphics feature of the Commodore 64 is activated using the CTRL|9 key or by printing CHR\$(18). This feature is turned off using the CTRL|0 key or by printing CHR\$(146) to the screen. These ASCII codes were determined in step 8 of the discovery exercises. You noticed there that the keys with 9 and 0 on them had RVS/ON and RVS/OFF on their fronts as a quick reference.

The reverse character feature can be used to print a reverse character space or square on the screen. For example, the direct mode line

```
PRINT CHR$(18);" "
```

will print a square (reverse character space) on the screen. Since the reverse character mode is turned off at the end of each print line, there is no need here to turn the reverse character feature off.

To create a blinking square, however, you must turn the reverse character feature on and off. A blinking square is created by typing a square, moving back a space using cursor left, CHR\$(157), and then printing a standard space. For example, the program

```
10 PRINT CHR$(18); " ";
20 PRINT CHR$(157);
30 PRINT CHR$(146); " ";
40 PRINT CHR$(157);
50 GOTO 10
60 END
```

causes a blinking square to appear on the screen when it is run. Lines 20 and 40 cause the cursor to move back the one space it moves ahead when the square or space is printed. Line 30 turns the reverse character feature off and then prints a space. Notice the semicolons at the end of lines 10 through 40 to prevent the cursor from moving down a line.

The Commodore 64 has a number of graphics characters that can be used to display diagrams and figures. The graphics characters appear in pairs on the fronts of some of the keys. In the start-up upper case/graphics mode, these characters are accessed in the following way. The characters on the left are accessed using the **C=** key. The characters on the right are accessed using the shift key.

Each graphics key has an ASCII code associated with it. A list of these ASCII codes can be found in Appendix F of the *Commodore 64 User's Guide*. Appendix F also has the ASCII codes for switching to the upper/lower case mode and to the upper case/graphics mode as well as the cursor movement and color codes.

### Controlling Color

The color used in printing a character on the blue background is controlled by the CTRL|1 through CTRL|8 keys. The actual color associated with each key appears on the fronts of the keys. The table below gives each color, and its associated CTRL key and ASCII code.

COLOR	CTRL KEY	ASCII CODE
Black	1	144
White	2	5
Red	3	28
Cyan	4	159
Purple	5	156
Green	6	30
Blue	7	31
Yellow	8	158

The start-up color combination is obtained by pressing **RUN/STOP|RESTORE**. This color combination is different from printing the characters in white.

The following direct mode statement prints a green square on the screen.

```
PRINT CHR$(18);CHR$(30);" "
```

If this statement is entered, after the green square is printed to the screen, all characters typed on the screen from the keyboard will be green on a blue background.

Additional information about changing the background and border colors as well as eight other colors available on the Commodore 64 can be found in Chapter 5 of the *Commodore 64 User's Guide*.

## 10-4 WORKING WITH PROGRAM EXAMPLES

### Example 1 - Cursor Movement

This program allows you to move a square about the screen with the U, D, R, and L keys. The same cursor placement subroutine you used in the Discovery Exercises is used here. Recall that printing **CHR\$(147)** clears the screen, **CHR\$(18)** turns on the reverse character feature, **CHR\$(146)** turns off the reverse character feature, and

CHR\$(157) causes the cursor to move back one space. The complete program follows.

```

100 PRINT CHR$(147)
110 LET R=13
120 LET C=20
130 GOSUB 1000
140 PRINT CHR$(18); " ";
150 GET A$
160 IF A$="" THEN 150
170 IF A$="U" THEN R=R-1
180 IF A$="D" THEN R=R+1
190 IF A$="R" THEN C=C+1
200 IF A$="L" THEN C=C-1
210 IF R<1 OR R>25 THEN 100
220 IF C<1 OR C>40 THEN 100
230 PRINT CHR$(157);CHR$(146); " ";
240 GOSUB 1000
250 PRINT CHR$(18); " ";
260 GOTO 150
270 END
1000 REM CURSOR PLACEMENT SUBROUTINE
1010 IF R<1 OR R>25 OR C<1 OR C>40
THEN END
1020 IF R=25 AND C=40 THEN END
1030 PRINT CHR$(19);
1040 IF R=1 THEN 1080
1050 FOR J=1 TO R-1
1060 PRINT CHR$(17);
1070 NEXT J
1080 IF C=1 THEN 1120
1090 FOR K=1 TO C-1
1100 PRINT CHR$(29);
1110 NEXT K
1120 RETURN

```

Lines 100 through 140 erase the screen and place a square in the center of the screen in row 13, column 20. Lines 150 and 160 check to see if a key has been pressed. Lines 170 through 200 determine the position where the square will be printed by changing the values of R and C. Line 230 erases the current square and lines 240 and 250 cause a new square to be printed at the new location.

Deleting line 230 where the current square is erased allows you to use this program to draw lines and shapes. You may enjoy running this program in both its forms.

### Example 2 - Bar Graphs

The following program will read the name and a numerical quiz grade for each person in a class. The names will be displayed on the screen with a bar graph to the right starting at tab position 10. The number of squares in the bar graph is equal to the numerical grade. The complete program follows.

```

100 PRINT CHR$(147)
110 READ N
120 FOR J=1 TO N
130 READ NAMES$,GRADE
140 PRINT NAMES$;TAB(10);
150 REM DRAW BAR GRAPH
160 FOR K=1 TO GRADE
170 PRINT CHR$(18);" ";
180 NEXT K
190 PRINT
200 PRINT
210 NEXT J
220 DATA 3
230 DATA JACK,4
240 DATA SUE,7
250 DATA RUTH,8
260 END

```

### Example 3 - Color Bar Graphs

The bar graphs program can be enhanced to draw the bar graphs in color. You do this by adding lines 30-90, 165, and 215 and by changing lines 130, 170, and 230-250. The complete program follows.

```

30 REM CREATE THE COLOR CODE ARRAY C
40 FOR I=1 TO 3
50 READ C(I)
60 NEXT I
70 REM GREEN,RED,YELLOW COLOR CODES
80 DATA 30,28,158

```

```

90 REM CLEAR THE SCREEN
100 PRINT CHR$(147)
110 READ N
120 FOR J=1 TO N
130 READ NAME$,GRADE,COLR
140 PRINT NAME$;TAB(10);
150 REM DRAW BAR GRAPH
160 FOR K=1 TO GRADE
165 REM C(COLR) IS 28,30, OR 158
170 PRINT CHR$(C(COLR));CHR$(18);" ";
CHR$(154);
180 NEXT K
190 PRINT
200 PRINT
210 NEXT J
215 REM NAME, GRADE AND COLR(1,2,3) DATA
220 DATA 3
230 DATA JACK,4,1
240 DATA SUE,7,2
250 DATA RUTH,8,3
260 END



```

Lines 30-70 store the color code numbers given in line 80 in the numerical array C. C(1) is the color code for green; C(2) for red; and C(3) for yellow. In line 170, the color code for light blue, the start-up color, is printed (CHR\$(154)) after each block. Thus, the names are always printed in the standard color. The color information (1,2,3) is added to lines 230-250. This program can be changed to create different bar graphs by changing or adding to the DATA statements that start at line 220.

#### Example 4 - Animation

This program causes a box to move across the screen as if it were animated. You will find the graphics characters for the corners on the A, S, Z, and X keys and the graphics character for one short line on the C key. The corners are found on the left side of the key so they are accessed with the Commodore (C=) key. The short line is accessed with the SHIFT key in the standard (upper case/graphics) mode. The short vertical line is typed by holding down the SHIFT key and pressing **B**. The part of line 160 in quotes is typed by holding down the Commodore key and pressing **A**, holding down



the SHIFT key and pressing  3 times and holding down the Commodore key and pressing . The complete program follows. On your screen, the lines will not be dashed.

```

100 PRINT CHR$(147);
110 LET J=1
120 PRINT CHR$(19);
130 FOR I=1 TO 5
140 PRINT
150 NEXT I
160 PRINT TAB(J);"  "
170 PRINT TAB(J);"  "
180 PRINT TAB(J);"  "
190 PRINT TAB(J);"  "
200 LET J=J+1
210 IF J>33 THEN END
220 PRINT CHR$(147);
230 GOTO 120
240 END

```

To begin with, line 100 erases the screen. Lines 120 through 150 move the cursor down 5 lines. Lines 160 through 190 print the box starting at column J which is initially set to 1 in line 110. Line 200 increases the value of J by 1, and line 220 checks to make sure J is not too large to print the box correctly. Lines 220 and 230 erase the screen and the box and start the process all over again. Other shapes could be created in lines 160 through 190 using the graphics characters. You may wish to create your own shape and animate it by running the program.

### 10-5 PROBLEMS

1. What will be printed by following program when it is run?

```

100 LET T=2
110 LET S=7
120 IF T<8 AND T>4 THEN PRINT "T OK":
GOTO 140
130 IF S<1 OR S>5 THEN PRINT "S OK"
140 END

```

2. What will happen if you run the following program?

```
10 LET A=5
20 PRINT "A=5 IS ";A=5
30 PRINT "A=6 IS ";A=6
40 END
```

3. How can you modify the cursor placement subroutine in step 10 of the Discovery Exercises to print the error message "OFF THE SCREEN" when the values of **R** and **C** are too large or too small?
4. Write a program that causes a green box to move repeatedly from the top of the screen to the bottom of the screen.
5. Write a program that causes a red box to move diagonally across the screen. The box should stop before it reaches the bottom of the screen.
6. Modify the data statements in the Color Bar Graphs program so that the first bar graph is yellow, the second is green, and the third is red.

### 10-6 PRACTICE TEST

1. What value of **B** must be entered for the following program to print OK?

```
10 INPUT B
20 IF B<7 AND B>5 THEN PRINT "OK"
30 END
```

---

2. What will happen when the direct mode statement below is executed?

```
PRINT CHR$(147);
```

---

3. What will be printed by the direct mode statement below?

```
PRINT CHR$(18); "J"
```

---

4. What color square will be printed by the following direct mode statement?

```
PRINT CHR$(28);CHR$(18); " "
```

---

5. What do you do to print the graphics character **┐** on the screen? (Hint: Look at the fronts of the keys.)
-



# RANDOM NUMBERS AND SIMULATIONS

## 11-1 OBJECTIVES

One of the most interesting applications of computers is the simulation of events or processes that involve an element of chance. For example, you might use a computer to simulate gambling games or perhaps to investigate how many bank tellers are required to ensure that arriving customers do not have to wait more than a few minutes for service. In this chapter you will see how the computer handles problems of this type. The objectives are as follows.

### **Understanding Random-Number Generators**

The random-number generator function of your Commodore 64 computer is the heart of all programs involving the element of chance or randomness. You will learn how to use these random-number generators in BASIC programs.

### **Designing Sets of Random Numbers**

Generally, the random-number generator is used to produce sets of random numbers with characteristics specified by the programmer. You will see how to generate any desired set of numbers.

### **Working with Program Examples**

The programming exercises and problems in this chapter use simulations and applications that involve the element of chance.

## 11-2 DISCOVERY EXERCISES

### Setting Up the Random-Number Generator

Before beginning the computer work, you need to know some important characteristics of random-number generators. By their very nature, these generators produce sequences of numbers that appear to have no pattern or relationship. For a random-number generator to be useful, it must return a different sequence of numbers each time you run a program that uses it. But suppose a program that uses random numbers is not working correctly. If the problem lies with the random numbers, it might be extremely difficult to correct, since different random numbers are generated each time the program is run. Consequently, Commodore 64 BASIC allows a sequence of random numbers to be repeated each time the program is run. To use this feature, you can give a starting number or seed for the random number generator. Be sure to use the feature only when you are troubleshooting a program.

1. Turn on the TV set and the computer. Unless otherwise specified, assume that different sequences of numbers are to be generated each time a program is run.
2. Type in the following program.

```
100 FOR I=1 TO 8
110 PRINT RND(1)
120 NEXT I
130 END
```

Run the program. How many decimal places are in the longest number?

---

Run the program again. Are the eight numbers generated the same as before?

---

3. Add line 90 as follows:

```
90 PRINT RND(-1)
```

Run the program twice. Are the sets of numbers the same?

---

PRINT RND(-1) starts the random-number generator at the number 2.99196472E-08 and prints it. The next eight numbers generated are based on this number. Thus, both sets of numbers are the same.

4. Clear the program in memory. Enter the following program:

```
100 LET L=.5
110 LET S=.5
120 FOR I=1 TO 100
130 LET X=RND(1)
140 IF X>L THEN 170
150 IF X<S THEN 190
160 GOTO 200
170 LET L=X
180 GOTO 200
190 LET S=X
200 NEXT I
210 PRINT "LARGEST = ";L
220 PRINT "SMALLEST = ";S
230 END
```

This program examines all the numbers generated by the RND function and keeps track of the largest and smallest numbers generated. As the program stands, it will generate 100 random numbers. Run the program. Wait several seconds and then record what was typed out.

---

5. Edit line 120 as follows.

```
120 FOR I=1 TO 1000
```

Now the program will generate 1000 random numbers. The program will take about 15 seconds to run. Run the program and record what was printed out.

---

From what you have seen thus far, what do you believe is the largest number the RND function will generate?

---

What about the smallest?

---

6. Now let's go on to some other ideas associated with random numbers. Clear the program in memory and enter the following program:

```
100 FOR I=1 TO 8  
110 PRINT INT(2*RND(1));  
120 NEXT I  
130 END
```

Run the program and record the output.

---

What were the only two numbers in the output?

---

7. Edit line 110 as follows.

```
110 PRINT INT(3*RND(1));
```

Display the program. If this program is run, what numbers do you think will be typed out?

---



Run the program several times. Can you predict anything about the sequence or pattern in which the numbers will appear?

---

8. Edit line 110 as follows.

```
110 PRINT INT(2*RND(1)+1);
```

What do you think the program will do now?

---

Run the program and record the output.

---

9. Edit line 110 as follows.

```
110 PRINT INT(4*RND(1)+5);
```

If the program is run, what do you think will be printed?

---

Run the program several times and describe the output.

---

Do you think there should be any pattern to the output?

---

10. Edit line 110 as follows. Note the missing semicolon.

```
110 PRINT INT(30*RND(1))/10
```

Display the program and study it carefully. What do you think this program will print?

---

Run the program and describe the printout.

---

11. Edit line 110 as follows.

```
110 PRINT INT(200*RND(1))/100
```

Display the program in memory. What do you think will happen if this program is run?

---

See if you were right. Run the program and record the output below.

---

12. Add line 90 as follows:

```
90 PRINT RND(-1)
```

Run the program several times. Are the sets of numbers the same?

---

13. This terminates the computer work for now. Turn off the TV set and the computer.

## 11-3 DISCUSSION

### Understanding Random-Number Generators

You need not be concerned with the details of how random numbers are generated. It is enough to say that several mathematical methods produce these numbers. The random-number generator is called on with the RND function. This function is used like other built-in functions in BASIC but differs in one important respect. The difference is that there seems to be no pattern or rule used to generate these numbers. Of course, this is precisely the point of the function. RND stands for "random." RND(1) generates numbers between 0 and 1 at random. All the numbers in the interval have an equal chance of showing up. (Actually, the range of numbers generated is

from 0.000000000 to 0.999999999. Occasionally, 0 shows up but the number 1 never does.) Other positive arguments for the RND function generate the same random numbers that the argument 1 does. The RND function also takes negative arguments. A negative argument yields the same number (between 0 and 1) each time it is used. A negative argument for the RND function used in a PRINT statement reseeds the random number generator by determining the starting number of the random numbers to be generated by RND(1).

■ **RND generates numbers in the range 0 to 0.999999999.**

**Designing Sets of Random Numbers**

Most often you do not want random numbers in the range produced by the RND function. You might want random integers in a certain range or a set of random numbers with a particular set of characteristics. Therefore, you need to know how to generate sets of random numbers with characteristics you can specify.

Let's begin with the characteristics of the RND function. RND(1) delivers numbers in the range 0 to 1, that is, from 0 to slightly less than 1. If you multiply RND(1) by N, you multiply the range of the function by N. Thus,  $N * \text{RND}(1)$  will produce random numbers in the range 0 to N. If desired, you could shift the numbers (keeping the same range) by adding a number.  $N * \text{RND}(1) + A$  would produce random numbers over the range A to (A+N) or from A to slightly less than (A+N). Finally, you could use the INT function to take the integer part of an expression and in this way generate random integers. The examples below indicate how you might use the RND(1) function.

BASIC Expression	Result
$5 * \text{RND}(1) + 10$	Random numbers in the range 10 to 15
$\text{INT}(5 * \text{RND}(1) + 10)$	Random integers 10, 11, 12, 13, 14
$\text{INT}(2 * \text{RND}(1) + 1)$	Random integers 1, 2
$100 * \text{RND}(1)$	Random numbers in the range 0 to 100

You may have encountered the notion of mean and standard deviation (see problem 9 in Chapter 9). You can use the RND function to generate numbers that appear to be drawn from a collection

of numbers having a given mean and standard deviation. The rule for generating these numbers is

$$X = M + S((\text{sum of 12 numbers from RND(1) function}) - 6)$$

where M and S are the desired mean and standard deviation, respectively. A subroutine is very useful in this application. As defined above, the values of X will appear to be from a collection of numbers with mean M and standard deviation S. The values of X can be used to simulate a process following a bell curve.

A note on troubleshooting: In Commodore 64 BASIC, there is a way to run a program several times and repeat the sequence of random numbers generated by the RND function (see step 12 of the Discovery Exercises). It is a good practice to write programs initially so that they will generate the same sequence of random numbers each time they are run. Once you are sure that the program is working correctly, you can modify the program to generate truly random numbers.

## 11-4 WORKING WITH PROGRAM EXAMPLES

### Example 1 - Flipping Coins

One of the easiest applications of random numbers is a coin-tossing simulation. Suppose you want to write a program that will produce the following printout:

TOSS	OUTCOME
1	H
2	T
3	T
4	H
(etc.)	

The outcome is to be determined randomly for each toss of the coin, with both heads and tails having equal probability. The program should print out the results of 10 coin tosses.

The first part of the program generates the column heads and the space below the heads.

```
100 PRINT "TOSS", "OUTCOME"
110 PRINT
```

Now open the loop to generate the 10 tosses of the coin.

```
120 FOR I=1 TO 10
```

The next step is to generate 0s and 1s randomly. Assume that 0 means a "heads" and 1 means "tails." Study the next program statement until you are sure it will produce 0s and 1s randomly.

```
130 LET X=INT(2*RND(1))
```

Now program the computer to analyze X to see whether a head (0) or a tail (1) has occurred.

```
140 IF X=0 THEN 170
150 PRINT I,"T"
160 GOTO 180
170 PRINT I,"H"
180 NEXT I
```

All that remains now is the END statement.

```
190 END
```

The complete program is listed below.

```
100 PRINT "TOSS","OUTCOME"
110 PRINT
120 FOR I=1 TO 10
130 LET X=INT(2*RND(1))
140 IF X=0 THEN 170
150 PRINT I,"T"
160 GOTO 180
170 PRINT I,"H"
180 NEXT I
190 END
```

Use this program to demonstrate how you can generate either different sequences of random numbers or identical sequences each time the program is run. Make the necessary changes in the program by adding a line 90 to make it return the same sequence again and again. See step 12 in the Discovery Exercises.

**Example 2 - Random Integers**

The next problem is to write a BASIC program to generate and print 50 random integers over the range 10 to 15. The only part of the program that requires much thought is the statement to generate the random integers, so concentrate on that statement.

Remember that RND(1) generates numbers over the range 0 to 1. Thus, 6\*RND(1) will generate numbers in the range 0 to 6. Actually the upper limit is 5.99999999. You use the integer function to convert from random numbers to random integers. INT(6\*RND(1)) will produce the integers 0, 1, 2, 3, 4, 5 randomly. Now it is clear that to get the desired numbers, you must add 10. Thus, the expression INT(6\*RND(1)) + 10 will produce the numbers you want.

Once you have this line, the program follows easily.

```
100 FOR I=1 TO 50
110 LET Y=INT(6*RND(1)) + 10
120 PRINT Y,
130 NEXT I
140 END
```

**Example 3 - Distribution of Random Numbers**

Suppose you generate a great number of integers at random over the range 1 to 10. If the random-number generator on the computer is working the way it should, you would expect to get the same number of each of the integers. If you generated 1000 integers, you would expect to get 100 1's, 100 2's, and so on. Your problem is to write a program to tally the random integers as they are generated and then print the totals. Then you can inspect these totals to judge how well the random-number generator works.

First, think about how to do the tally. A good way is to use a one-dimensional subscripted array. X(1) will contain the number of 1s generated, X(2) the number of 2s, and so forth, up to X(10). Thus, the first task is to dimension the array and set all the values in the array equal to 0.

```
100 DIM X(10)
110 FOR I=1 TO 10
120 X(I)=0
130 NEXT I
```

Next, open a loop to generate 1000 numbers, generate the random integers, then use the integers as subscripts to increment the appropriate counters in the array.

```

140 FOR I=1 TO 1000
150 LET Y=INT(10*RND(1)) + 1
160 LET X(Y)=X(Y) + 1
170 NEXT I

```

Now all that remains is to print out the contents of the array X.

```

180 FOR J=1 TO 10
190 PRINT J,X(J)
200 NEXT J
210 END

```

The complete program follows:

```

100 DIM X(10)
110 FOR I=1 TO 10
120 LET X(I)=0
130 NEXT I
140 FOR I=1 TO 1000
150 LET Y=INT(10*RND(1)) + 1
160 LET X(Y)=X(Y) + 1
170 NEXT I
180 FOR J=1 TO 10
190 PRINT J,X(J)
200 NEXT J
210 END

```

It might be interesting to run this program and see for yourself how well your random-number generator works. As you decrease the number of integers generated, the chances for even distribution of integers diminish. On the other hand, if you generate more random numbers, the distribution should be more equitable. The computer will take about 20 seconds to print the results.

#### **Example 4 - Random Walk**

This program simulates a random walk on a tiled floor. You walk aimlessly, or at random, from tile to tile. You may walk back over some tiles you have already walked on.

Line 150 assigns **A** an integer value randomly chosen from 1, 2, 3, and 4. Lines 160 through 200 use the value of **A** to decide whether the next tile you walk on is up, down, right, or left. Line 230 "walks" on the tile by printing a square. Lines 200 and 210 start the process over again if you walk off the tiled floor. Once again the cursor placement subroutine starting in line 1000 is used to position the cursor prior to printing the square character.

```

100 PRINT CHR$(147)
110 LET R=13
120 LET C=20
130 GOSUB 1000
140 PRINT CHR$(18);" ";
150 LET A=INT(4*RND(1))+1
160 IF A=1 THEN R=R-1
170 IF A=2 THEN R=R+1
180 IF A=3 THEN C=C-1
190 IF A=4 THEN C=C+1
200 IF R<1 OR R>25 THEN 100
210 IF C<1 OR C>40 THEN 100
220 GOSUB 1000
230 PRINT CHR$(18);" ";
240 GOTO 150
250 END

1000 REM CURSOR PLACEMENT SUBROUTINE
1010 IF R<1 OR R>25 OR C<1 OR C>40
THEN END
1020 IF R=25 AND C=40 THEN END
1030 PRINT CHR$(19);
1040 IF R=1 THEN 1080
1050 FOR J=1 TO R-1
1060 PRINT CHR$(17);
1070 NEXT J
1080 IF C=1 THEN 1120
1090 FOR K=1 TO C-1
1100 PRINT CHR$(29);
1110 NEXT K
1120 RETURN

```



**Example 5 - Random Colors**

This program colors a rectangle in the center of the screen with squares of random color and position. The heart of the program is in lines 220, 230, and 240 where the integer valued functions defined in lines 180, 190, and 200 are used to determine the color and row and column position of a square. The functions are FNC, FNROW, and FNCOL. FNC returns a random integer chosen from the numbers 1 through 8. FNROW returns a random integer from 7 through 16, and FNCOL returns a random integer from 15 through 24. Line 270 prints the colored square on the screen. The program follows without the necessary cursor placement subroutine, which should start at line 1000.

```

100 REM CREATE THE COLOR ARRAY C
110 FOR I=1 TO 8
120 READ C(I)
130 NEXT I
140 DATA 144,5,28,159,156,30,31,158
150 REM CLEAR THE SCREEN
160 PRINT CHR$(147);
170 REM DEFINE THE COLOR, ROW AND COLUMN
    RANDOM NUMBER FUNCTIONS
180 DEF FNC(I)=INT(8*RND(1))+1
190 DEF FNROW(I)=INT(10*RND(1))+7
200 DEF FNCOL(I)=INT(10*RND(1))+15
210 REM SELECT A RANDOM COLOR, ROW,
    AND COLUMN
220 LET COLR=C(FNC(1))
230 LET R=FNROW(1)
240 LET C=FNCOL(1)
250 GOSUB 1000
260 REM PRINT THE SQUARE
270 PRINT CHR$(COLR);CHR$(18);" ";
280 GOTO 210
290 END

```

**Example 6 - Birthday Pairs in a Crowd**

Suppose there are 50 people in a room. What is the probability that two of the people have the same birthday? (Consider only the day of the year, not the year of birth.) This problem is famous in probability theory and has surprising results. You can attack the

problem with the following strategy. By generating random integers over the range 1 to 365, you can simulate a birthday for each of the strangers. If you use a one-dimensional array for the birthdays as they are generated, it is easy to check for identical birthdays. Beginning with the first birthday  $B(1)$ , program the computer to check if it matches any of the remaining ones. Then have the computer check  $B(2)$  in the same way, and so on.

In this example, first look at the complete program, then go back and figure out what is taking place in each line.

```

100 DIM B(50)
110 FOR I=1 TO 50
120 LET B(I)=INT(365*RND(1)) + 1
130 NEXT I
140 LET F=0
150 FOR I=1 TO 49
160 FOR J=I+1 TO 50
170 IF B(I) <> B(J) THEN 190
180 LET F=F+1
190 NEXT J
200 NEXT I
210 PRINT "NUMBER OF BIRTHDAY PAIRS IS";F
220 END

```

Of course, line 100 merely dimensions an array for 50 elements. Lines 110 through 130 load the array with random integers selected over the range 1 to 365 inclusive. In line 140, you set the variable  $F$  equal to 0. This variable will keep track of the number of birthdays to be compared with the rest of the birthdays in the list. Since there must be at least one birthday left in the list against which to compare, the value of  $I$  stops at 49. Line 160 sets up the second half of the comparison.

$J$  begins at the next value past the current value of  $I$  and runs through the rest of the list. The test for a birthday pair is made in line 170. If no match is found, the computer jumps to the next value of  $J$ . If a match is found, the pair counter in line 180 is increased by 1. The results are printed in line 210. One problem with the program is that it would record three people having the same birthday as two birthday pairs. Can you figure out a way to fix this?

This is a very interesting program to experiment with. You can modify the number of people in the crowd by making simple changes in the program. You can run the program many times to

see how many birthday pairs, on average, there will be in a crowd of a specified size.

## 11—5 PROBLEMS

1. Write a program to generate and print 25 random numbers like 5.3 of the form X.Y where X and Y are digits selected randomly from the set 0, 1, 2, 3, ..., 9.
2. Write a program to generate and print 50 integers selected at random from the range 13 to 25.
3. What will be printed if the following program is run?

```
100 FOR N=1 TO 20
110 PRINT INT(20*RND(1))+1)/100
120 NEXT N
130 END
```

4. If you run the following program, what will you see on the screen?

```
100 FOR I=1 TO 10
110 PRINT INT(100*RND(1))/10
120 NEXT I
130 END
```

5. Write a program that will simulate tossing a coin, 10, 50, 500, and 1000 times. In each case, print the total number of heads and tails that occur.
6. Construct a dice-throwing simulation in BASIC. The dice are to be thrown 20 times. For each toss, print the dice faces that are uppermost.
7. Write a program to generate and print out the average of 100 random numbers selected from the range 0 to 1.
8. Modify the program from example 5 and run it as many times as needed to find the size of crowd such that there is a 50 percent chance that at least two people in the crowd have the same birthday.
9. John and Bill want to meet at the library. Each agrees to arrive at the library sometime between 1:00 and 2:00 p.m. They further agree that they will wait 10 minutes after arriving (but not after 2:00 p.m.), and if the other person has not arrived, will leave. Write a program to compute the probability that John and Bill will meet. Do a simulation of the problem using the random-number generator.

10. Suppose a basket contains colored golf balls. There are 10 red, 5 blue, 2 green, and 11 yellow balls. Write a program to simulate drawing five balls at random from the bucket. The balls are drawn in sequence and are not replaced after being drawn.
11. Use the rule given in the discussion section in this chapter to generate and print out 25 numbers selected at random from a bell curve distribution of numbers with mean 10 and standard deviation 2. Round off the numbers to two places past the decimal point.
12. Suppose a soap manufacturer decides to select a five character brand name. The first, third, and fifth characters are selected at random from the letters BCDFGHJKLMNPQRSTUVWXYZ. The second and fourth letters are selected at random from the vowels AEIOU. Write a program to generate and print out 100 trial soap names using the rules above.
13. Modify the program from problem 6 to simulate tossing a pair of dice 1000 times. Print out the number of times each of the 11 possible outcomes happened in the simulation.

### 11—6 PRACTICE TEST

1. Write a program to generate and print out 100 random integers selected from the set 1, 2, 3, and 4.
2. Write a program to generate and print 100 random numbers over the range 25 to 50.
3. What will be printed if you run the following program?

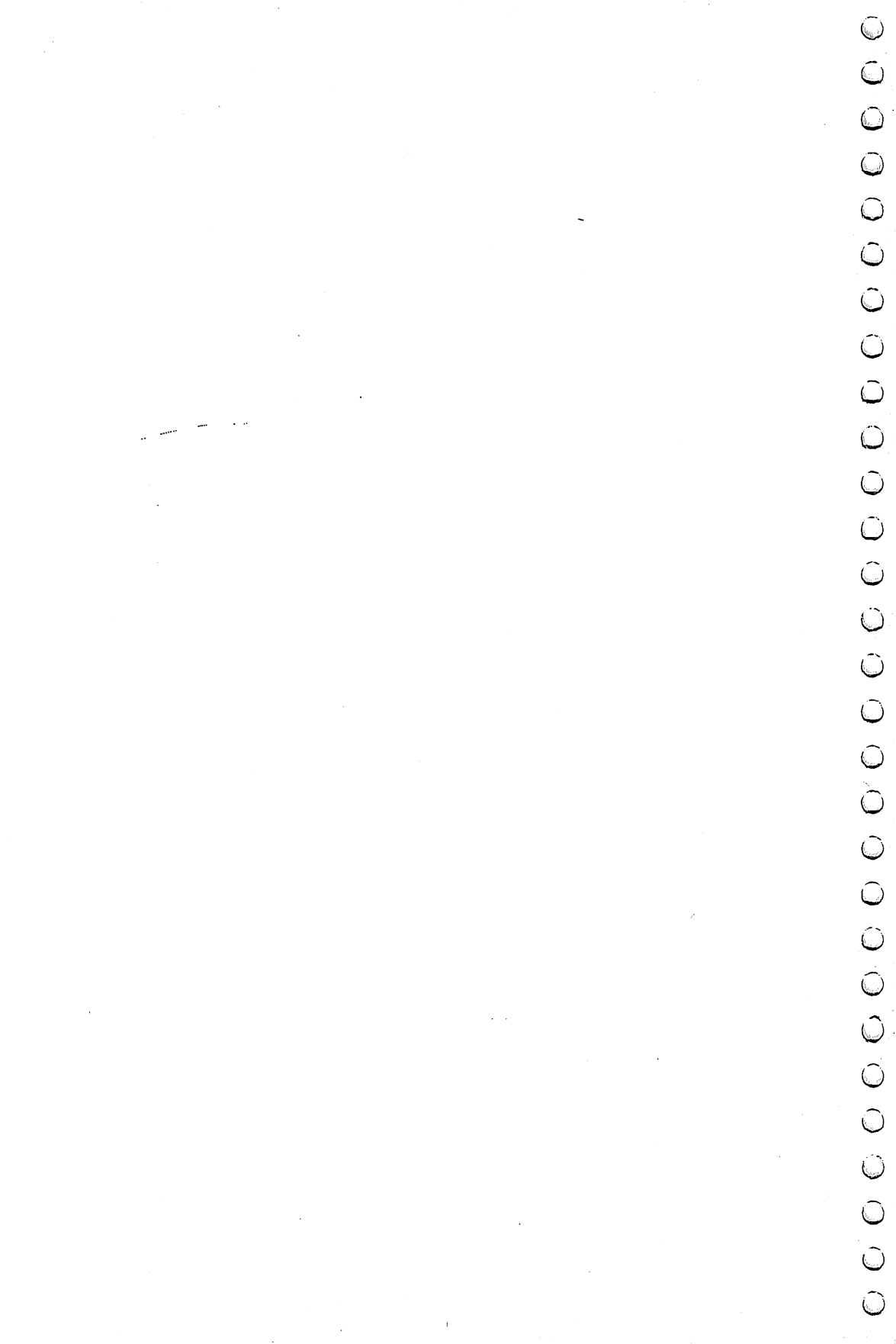
```
100 FOR I=1 TO 10
110 LET N=INT(2*RND(1)+1)
120 IF N=1 THEN 150
130 PRINT "WHITE"
140 GOTO 160
150 PRINT "RED"
160 NEXT I
170 END
```

---

4. What will be printed if you run the following program?

```
100 FOR J=1 TO 5
110 PRINT INT(1000*RND(1))/100
120 NEXT J
130 END
```

---



## CHAPTER 12

# FILES

### 12-1 OBJECTIVES

You can use files saved on diskette to store and retrieve collections of information. In this chapter you will learn to use files to manipulate collections of information.

#### **Opening and Closing Information Paths**

Before you can store information to a file, you have to create a path to use in transferring information from the computer to the disk drive. You will learn to create and dismantle these paths.

#### **Writing Information to a File**

You will learn to create files in which to store information.

#### **Retrieving Information from a File**

Once information is stored, you must be able to retrieve it. You will study methods of retrieving information stored on files.

#### **Working with Program Examples**

Frequently, you need to change the information stored on files. You will learn how to do this. You will also write programs that extract and format information stored on files.

### 12-2 DISCOVERY EXERCISES

Historically, files have been difficult to work with on computers. Fortunately, advances in computer languages have significantly eased the difficulties. You will find that writing programs that use files

requires many techniques and concepts learned in previous chapters.

1. Turn on your TV set, disk drive, and computer. Place a formatted disk in the drive. (See chapter 3.) Clear the memory.
2. Do not use ? as an abbreviation for PRINT in these exercises. Type the following program.

```

90 OPEN 15,8,15
100 OPEN 2,8,2,"GIFTLIST,L," + CHR$(50)
110 PRINT "NAME: ";
120 INPUT NAME$
130 PRINT "AMOUNT: ";
140 INPUT AMT
150 PRINT#15,"P"CHR$(2)CHR$(3)CHR$(0)CHR$(1)
160 PRINT#2,NAME$;",";AMT
170 CLOSE 2
180 CLOSE 15
190 END

```

A record is a structured set of information. In order to store a record on diskette, you must be able to transfer two types of information: information about placement of the record and the information in the record. Line 90 establishes a placement information path. Line 100 establishes a record information path. These two paths together will be called the information path. The 15s in lines 90, 150, and 180 refer to the placement information path. The 2s in lines 100, 150, 160, and 170 refer to the record information path. The 8 refers to the disk drive unit. The "GIFTLIST,L," + CHR\$(50) creates and names the file GIFTLIST with the length of each record set at 50. The 3 in line 150 indicates that you are going to place information in record 3 of the file. Display the program and check it, especially the semicolons and commas. Note that there is no space between PRINT and # in lines 150 and 160.

3. Run the program and enter your name and an appropriate gift amount. Did the disk whirr?

---

What happened to the information represented by NAME\$ and AMT?

---



4. Save this program as WRITEDISK. (See chapter 3.)
5. Type

```
LOAD"$",8  
LIST
```

to obtain a list of files and programs on your diskette. Commodore 64 BASIC programs are identified on the list by a PRG in the third column. What letters are in the third column for the file GIFTLIST?

---

REL stands for relative, the technical name for the type of file you are using.

6. Clear the memory and type in the following program.

```
90 OPEN 15,8,15  
100 OPEN 2,8,2,"GIFTLIST,L," + CHR$(50)  
110 PRINT#15,"P"CHR$(2)CHR$(3)CHR$(0)CHR$(1)  
120 INPUT#2,NAME$,AMT  
130 CLOSE 2  
140 CLOSE 15  
150 END
```

Display the program and check it. Run the program. Did the disk whirr?

---

Was a question mark placed on the screen by line 120?

---

Was anything displayed on the screen by the program?

---

7. Add line 145 as follows

```
145 PRINT NAME$,AMT
```

Run the program. In line 110, you use the PRINT#15 statement to pass the placement ("P") information. The disk drive locates the GIFTLIST file (CHR\$(2)), record 3 (CHR\$(3)). The information is read from the record into the variables NAME\$ and AMT in line 120. Is that enough to make the information useful?

---

8. Save this program as READDISK.
9. You must open and close an information path each time you use it. To write to a file, use a PRINT# statement to locate the correct record and a PRINT# statement to transfer the information into the record on diskette. To read from a file, use a PRINT# statement to locate the correct record and an INPUT# statement to transfer the information in that record to variables in memory.
10. Now move on to using files with more than one record. Load the program WRITEDISK. Display the program. Add the following lines.

```
80 LET I=1
105 PRINT "TYPE QUIT FOR NAME WHEN DONE"
125 IF NAME$="QUIT" THEN CLOSE 2:CLOSE 15:END
183 LET I=I + 1
185 GOTO 90
```

11. Change line 150 to read

```
150 PRINT#15,"P"CHR$(2)CHR$(I)CHR$(0)CHR$(1)
```

Display the program once again to check its accuracy. The program should now look like this:

```
80 LET I=1
90 OPEN 15,8,15
100 OPEN 2,8,2,"GIFTLIST,L," + CHR$(50)
105 PRINT "TYPE QUIT FOR NAME WHEN DONE"
110 PRINT "NAME: ";
```

```

120 INPUT NAMES$
125 IF NAME$= "QUIT" THEN CLOSE 2:CLOSE 15:END
130 PRINT "AMOUNT: ";
140 INPUT AMT
150 PRINT#15,"P"CHR$(2)CHR$(1)CHR$(0)CHR$(1)
160 PRINT#2,NAMES$;",";AMT
170 CLOSE 2
180 CLOSE 15
183 LET I=I + 1
185 GOTO 90
190 END

```

Run the program. At the input prompt, enter the following names and amounts:

```

ANN 150
BOB 35
JACK 75
SUE 45

```

Notice that the amounts are above and below 50. What do you type to end the information entry?

---

Try it to see if you were correct.

12. Save this new program under the same name WRITEDISK by typing

```
SAVE "@@:WRITEDISK",8
```

13. Load READDISK. Display the program. Delete lines 130, 140, 145, and 150. Add the following lines.

```

80 LET I=1
135 PRINT NAMES$,AMT
137 LET I=I + 1
139 GOTO 110
140 CLOSE 2
150 CLOSE 15
160 END

```

14. Change line 110 to read

```
110 PRINT#15,"P"CHR$(2)CHR$(1)CHR$(0)CHR$(1)
```

The program should look like this:

```
80 LET I=1
90 OPEN 15,8,15
100 OPEN 2,8,2,"GIFTLIST,L," + CHR$(50)
110 PRINT#15,"P"CHR$(2)CHR$(1)CHR$(0)CHR$(1)
120 INPUT#2,NAME$,AMT
135 PRINT NAME$,AMT
137 LET I=I + 1
139 GOTO 110
140 CLOSE 2
150 CLOSE 15
160 END
```

Check the program to make sure all the punctuation is correct. What do you think will happen if you run the program?

---

Run the program and see if you were right.

15. Press RUN/STOP|RESTORE. You can avoid the red blinking light error on the disk drive by adding line 115 and the subroutine in lines 500-530 below.

```
115 GOSUB 500
500 REM CHECK FOR DISK ERROR SUBROUTINE
510 INPUT#15,ERRNUM
520 IF ERRNUM=50 THEN 140
530 RETURN
```

Run the program. Did a red blinking light disk error occur this time?

---

16. Now extract some information from your file. Display the program. Add the following line.

```
134 IF AMT<50 THEN 137
```

Run the program. Were all the names printed out?

---

17. You can also compile information from the file. Display the program. Change lines 160 and 134 and add lines 70 and 170 as follows

```
160 PRINT "THERE ARE ";C;"GIFTS OVER 50"
134 IF AMT>50 THEN LET C=C + 1
70 LET C=0
170 END
```

Display the program. Your program should look like this:

```
70 LET C=0
80 LET I=1
90 OPEN 15,8,15
100 OPEN 2,8,2,"GIFTLIST,L," + CHR$(50)
110 PRINT#15,"P"CHR$(2)CHR$(I)CHR$(0)CHR$(1)
115 GOSUB 500
120 INPUT#2,NAME$,AMT
134 IF AMT>50 THEN LET C=C + 1
135 PRINT NAME$,AMT
137 LET I=I + 1
139 GOTO 110
140 CLOSE 2
150 CLOSE 15
160 PRINT "THERE ARE ";C;"GIFTS OVER 50"
170 END
500 REM CHECK FOR DISK ERROR SUBROUTINE
510 INPUT#15,ERRNUM
520 IF ERRNUM=50 THEN 140
530 RETURN
```

When the program is run, what do you think will be displayed?

---

Run the program. Does the display agree with the information you wrote to the file?

---

You may wish to display the program and study it.

18. Now add up some things. Display the program. Delete line 160. Change lines 70, 134, and 150 as follows.

```
70 LET SUM=0
134 LET SUM=SUM + AMT
150 PRINT "TOTAL GIFT SUM IS ";SUM
```

Display the program. Run the program. What was the total of the gifts in the information you entered?

---

Delete line 135. Run the program. Is the information in each record printed out this time?

---

19. That ends the Discovery Exercises. Turn off the computer and the TV set. Remove the diskette and turn off the disk drive. Go on to the next section.

## 12-3 DISCUSSION

### Opening and Closing Information Paths

In using a diskette file, you worked with information paths. These paths have two parts: (1) for information about the placement of a record and (2) for information in the record. They are needed to transfer records from the computer to the disk drive and back. The placement information path is always established with OPEN 15,8,15. OPEN and CLOSE statements initiate and terminate information paths. For example, the program lines

```
200 OPEN 15,8,15
300 OPEN 2,8,2,"FILEONE,L," + CHR$(70)
400 CLOSE 2
500 CLOSE 15
```

open and close the two parts of the information path for the file FILEONE. Line 200 opens the placement information path while line 300 opens the record information path for the file FILEONE. Line 300 creates the file, if necessary, as a relative (REL) file, the type of file we have used in the Discovery Exercises. Also, in line 300, each record in FILEONE is given a length 70 by the L and CHR\$(70). Lines 400 and 500 close the information path for FILEONE by closing its paths 2 and 15.

### Writing Information to a File

Once an information path to a file is opened, you can write to the file or read from it. To write to a file, you use a PRINT# statement with four CHR\$ functions to locate the record to which you wish to write. For example,

```
110 PRINT#15,"P"CHR$(2)CHR$(3)CHR$(0)CHR$(1)
```

uses the placement information path to locate the file associated with record information path 2 and places ("P") the file at record 3. The CHR\$(1) at the end of line 110 indicates that you wish to access record 3 starting at position 1. The third CHR\$ function is used only if more than 255 records need to be accessed. See page 35 of the *VIC-1541 Single Drive Floppy Disk User's Manual* for further information.

To write information to the file, another PRINT# statement must be used. This PRINT# statement uses the record information path 2. For example, the pair of lines

```
110 PRINT#15,"P"CHR$(2)CHR$(3)CHR$(0)CHR$(1)
120 PRINT#2,FIRSTNAME$;",";GRADE
```

will place the value of string variable FIRSTNAME\$ and the value of numeric variable GRADE in the third record of the file associated with record information path 2 starting again at position 1. Line 120 also places a comma between these two values as a separator. INPUT# statements use these separators to decide where variables end.

In the following example, an information path is initiated to a file named QUIZ1 and allows some information to be written to this file.

```

90 LET I=1
100 OPEN 15,8,15
110 OPEN 4,8,4,"QUIZ1,L," + CHR$(50)
120 PRINT "FIRST NAME: ";
130 INPUT FIRSTNAME$
140 IF FIRSTNAME$="QUIT" THEN CLOSE 4:
CLOSE 15:END
150 PRINT "LETTER GRADE: ";
160 INPUT GRADE$
170 PRINT#15,"P"CHR$(4)CHR$(I)CHR$(0)CHR$(1)
180 PRINT#4,FIRSTNAME$;",";GRADE$
190 LET I=I + 1
200 CLOSE 4
210 CLOSE 15
220 GOTO 100
230 END

```

In line 100 and 110, the information path to QUIZ1 is established. (file QUIZ1 is created if necessary) and each record is assigned a length of 50 characters. In line 170, record I is located. In line 180, **FIRSTNAME\$**, a comma, and **GRADE\$** are written to record I of the file associated with record information path 4 (QUIZ1). Line 210 and 220 terminate the information path.

■ Use two PRINT# statements to write to a file.

**Retrieving Information from a File**

To read information from a file use a PRINT# and an INPUT# statement, as in the following example.

```

90 LET I=1
100 OPEN 15,8,15
110 OPEN 6,8,6,"QUIZ1,L," + CHR$(50)
120 PRINT#15,"P"CHR$(6)CHR$(I)CHR$(0)CHR$(1)
130 INPUT#15,ERRNUM
140 IF ERRNUM=50 THEN CLOSE 6:CLOSE 15:END
150 INPUT#6,FIRSTNAME$,GRADE$
160 PRINT FIRSTNAME$,GRADE$
170 LET I=I + 1
180 GOTO 120
190 END

```



Line 120 is similar to line 170 in the previous program. It locates the file associated with record information path 6 (QUIZ1) and places that file at record I, position 1. Recall that without the instruction in line 160, the computer would display nothing on the screen. The INPUT# statement in line 150 only reads information into the variables. What is then done with the variables depends on the desired outcome. Again, PRINT# and INPUT# are a pair.

■ **Use PRINT# and INPUT# statements to read a file.**

An error number is recorded on the placement information path (15) for each disk access. You can read the error number with an INPUT#15 statement. If no error occurs, the error number is 0. In line 130, you look at the placement information path to find the error number (ERRNUM) of the last disk access. In line 140, you determine whether the error number is 50 (the RECORD NOT PRESENT error). If the record is present, then you have reached the end of the file of records and the two parts of the information path are closed.

More elaborate relative file structuring techniques and error handling techniques are in the *VIC-1541 Single Drive Floppy Disk User's Manual*. You will explore some of these techniques.

Another kind of file, called a sequential file, is also available in Commodore 64 BASIC. You have seen that relative (also called direct access) files allow you to read from and write to any record by indicating the number of the record you want to use. By contrast, to access information in a sequential file, you read each and every item in the file until you reach the information you want. Reading sequential files is much like using the READ statement with information stored in a program in DATA statements. Certain tasks are better handled with sequential files than with relative files. Information about sequential files can be found in your *VIC-1541 Single Drive Floppy Disk User's Manual*.

## **12-4 PROGRAM EXAMPLES**

You will write a set of programs that can be used to create and maintain a mailing list for advertising and billing purposes.

**Example 1 - Mail List Data Entry Program**

A mailing list contains names, addresses, and other information about individuals. Thus, the program might request the following items.

FIRST NAME: (You type in)  
 LAST NAME: (You type in)  
 STREET: (You type in)  
 CITY: (You type in)  
 ZIP CODE: (You type in)  
 BALANCE: (You type in)

The program will request this information for every person to be placed on the mailing list. You can program the computer to break the INPUT loop when QUIT is entered for the first name. The information would then be stored to a relative file called MAILLIST. In line 1020, ERRNUM = 0 checks for the OK (no error) error.

The complete program follows.

```

100 LET I=2
110 OPEN 15,8,15 TELEPHONE NUMBERS
120 OPEN 2,8,2,"MAILLIST,L," + CHR$(120)
130 PRINT "FIRST NAME: ";
140 INPUT FIRST$
150 IF FIRST$="QUIT" THEN 310
160 PRINT "LAST NAME: ";
170 INPUT LAST$
180 PRINT "STREET: ";
190 INPUT STREET$
200 PRINT "CITY: ";
210 INPUT CITY$
220 PRINT "ZIP CODE: ";
230 INPUT ZIP$
240 PRINT "PAYMENT BALANCE: ";
250 INPUT BAL
260 PRINT#15,"P"CHR$(2)CHR$(I)CHR$(0)CHR$(1)
270 GOSUB 1000
280 PRINT#2,FIRST$,"";LAST$,"";STREET$;
  " ";CITY$,"";ZIP$,"";BAL
290 LET I=I + 1
300 GOTO 130

```

```

310 GOSUB 2000
320 CLOSE 2
330 CLOSE 15
340 END
1000 REM CHECK RECORD NOT PRESENT ERROR
1010 INPUT#5,ERRNUM,ERRNAME$
1020 IF ERRNUM=50 OR ERRNUM=0 THEN RETURN
1030 PRINT ERRNUM,ERRNAME$
1040 CLOSE 2
1050 CLOSE 15
1060 END
1070 RETURN
2000 REM PLACE NUMBER OF RECORDS AND
2010 REM RECORD LENGTH IN RECORD 1
2020 PRINT#15,"P"CHR$(2)CHR$(1)CHR$(0)CHR$(1)
2030 GOSUB 1000
2040 PRINT#2,I-1;",";120
2050 RETURN

```

Save this program as ENTERDATA. You will use it again.

### Example 2 - Add Records Program

This program allows you to add records to a given file. The program calls for a small modification of the previous program, ENTERDATA. The modification is simply the deletion of line 100 and the addition of lines 123 through 127 as shown below.

```

110 OPEN 15,8,15
120 OPEN 2,8,2,"MAILLIST,L," + CHR$(120)
123 PRINT#15,"P"CHR$(2)CHR$(1)CHR$(0)CHR$(1)
125 INPUT#2,RECNUM
127 LET I=RECNUM + 1
130 PRINT "FIRST NAME: ";
140 INPUT FIRST$
150 IF FIRST$="QUIT" THEN 310
160 PRINT "LAST NAME: ";
170 INPUT LAST$
180 PRINT "STREET: ";
190 INPUT STREET$
200 PRINT "CITY: ";
210 INPUT CITY$

```

```

220 PRINT "ZIP CODE: ";
230 INPUT ZIP$
240 PRINT "PAYMENT BALANCE: ";
250 INPUT BAL
260 PRINT#15,FIRST$;",";LAST$;",";STREET$;
CITY$;",";ZIP$;",";BAL
270 GOSUB 1000
280 PRINT#2,FIRST$;",";LAST$;",";STREET$;
",";CITY$;",";ZIP$;",";BAL
290 LET I=I + 1
300 GOTO 130
310 GOSUB 2000
320 CLOSE 2
330 CLOSE 15
340 END

1000 REM CHECK RECORD NOT PRESENT ERROR
1010 INPUT#15,ERRNUM,ERRNAME$
1020 IF ERRNUM=50 OR ERRNUM=0 THEN RETURN
1030 PRINT ERRNUM,ERRNAME$
1040 CLOSE 2
1050 CLOSE 15
1060 END
1070 RETURN

2000 REM PLACE NUMBER OF RECORDS AND
2010 REM RECORD LENGTH IN RECORD 1
2020 PRINT#15,"P"CHR$(2)CHR$(1)CHR$(0)CHR$(1)
2030 GOSUB 1000
2040 PRINT#2,I-1;",";120
2050 RETURN

```

Save this program as ADDRECORDS.

### Example 3 - Mailing Label Program

This program will use the MAILLIST relative file to generate mailing labels for envelopes. The labels should have three lines as shown in the following example.

```

GEORGE JONES
1234 DATAFILE DRIVE
SAN JOSE, CA 95009

```

The complete program follows.

```

100 LET I=2
110 OPEN 15,8,15
120 OPEN 2,8,2,"MAILLIST,L," + CHR$(120)
125 REM READ IN LAST RECORD NUMBER
130 PRINT#15,"P"CHR$(2)CHR$(1)CHR$(0)CHR$(1)
140 INPUT#2,RECNUM
145 REM READ IN RECORD
150 PRINT#15,"P"CHR$(2)CHR$(1)CHR$(0)CHR$(1)
160 INPUT#2,FIRST$,LAST$,STREET$,CITY$,
ZIP$,BAL
165 REM PRINT OUT A LABEL
170 PRINT FIRST$;" ";LAST$
180 PRINT STREET$
190 PRINT CITY$;" CA ";ZIP$
200 PRINT
210 PRINT
220 LET I=I + 1
230 IF I <= RECNUM THEN 150
240 CLOSE 2
250 CLOSE 15
260 END

```

Save this program as LABELS.

#### Example 4 - Selected Labels Program

Having written the previous program, you can easily select records in a file based on a specific condition. In this example, you obtain a set of labels for billing customers by selecting those customers whose balances are greater than zero. Modify the previous program simply by changing line 165 and adding line 167. The complete program reads as follows.

```

100 LET I=2
110 OPEN 15,8,15
120 OPEN 2,8,2,"MAILLIST,L," + CHR$(120)
125 REM READ IN LAST RECORD NUMBER
130 PRINT#15,"P"CHR$(2)CHR$(1)CHR$(0)CHR$(1)
140 INPUT#2,RECNUM

```

```

145 REM READ IN RECORD
150 PRINT#15,"P"CHR$(2)CHR$(1)CHR$(0)CHR$(1)
160 INPUT#2,FIRST$,LAST$,STREET$,CITY$,
ZIP$,BAL
165 REM DON'T PRINT LABEL IF BAL <= 0
167 IF BAL <= 0 THEN 220
170 PRINT FIRST$;",";LAST$
180 PRINT STREET$
190 PRINT CITY$;" CA ";ZIP$
200 PRINT
210 PRINT
220 LET I=I + 1
230 IF I <= RECNUM THEN 150
240 CLOSE 2
250 CLOSE 15
260 END

```

Save this program as SELECTEDLABELS.

#### Example 5 - Modifying the MAILLIST File

This program allows you to modify any of the items (for instance, the address) in a *record* (all the information about one person) in the MAILLIST file. To do this, the program must know which record and item you wish to change. The program should ask for a last name and then display all the items for a record where the **LAST\$** matches the last name you enter. If the record you wish to modify is displayed, you should then be asked to modify one or more items in the record. If not, the program should continue to display the other records you wish to modify. In line 290, PRINT CHR\$(147) clears the text screen.

The complete program follows.

```

100 OPEN 15,8,15
110 OPEN 2,8,2,"MAILLIST,L," + CHR$(120)
120 PRINT#15,"P"CHR$(2)CHR$(1)CHR$(0)CHR$(1)
130 GOSUB 2000
140 INPUT#2,RECNUM
150 PRINT "PLEASE ENTER THE LAST"
160 PRINT "NAME OF THE RECORD"
170 PRINT "YOU WISH TO MODIFY.";
180 PRINT "<ENTER QUIT TO END>"

```

```
190 INPUT NAMES$
200 IF NAMES$="QUIT" THEN CLOSE 2:
CLOSE 15:END
210 FOR J=2 TO RECNUM
220 PRINT#15,"P"CHR$(2)CHR$(J)CHR$(0)CHR$(1)
230 GOSUB 2000
240 INPUT#2,FIRST$,LAST$,STREET$,CITY$,
ZIP$,BAL
250 IF NAMES$=LAST$ THEN 290
260 NEXT J
270 PRINT "NO MATCH FOUND"
280 GOTO 150
290 PRINT CHR$(147)
300 PRINT "1. ";FIRST$
310 PRINT "2. ";LAST$
320 PRINT "3. ";STREET$
330 PRINT "4. ";CITY$
340 PRINT "5. ";ZIP$
350 PRINT "6. ";BAL
360 PRINT "7. EXIT"
370 PRINT "WHAT IS THE NUMBER OF THE"
380 PRINT "ITEM YOU WISH TO MODIFY ";
390 INPUT N
400 IF N<1 OR N>7 THEN 290
410 ON N GOSUB 500,600,700,800,900,1000,4000
420 GOTO 290
430 CLOSE 2
440 CLOSE 15
450 END
500 PRINT "FIRST NAME: ";
510 INPUT FIRST$
520 GOSUB 3000
530 RETURN
600 PRINT "LAST NAME: ";
610 INPUT LAST$
620 GOSUB 3000
630 RETURN
700 PRINT "STREET: ";
710 INPUT STREET$
720 GOSUB 3000
730 RETURN
```

```

800 PRINT "CITY: ";
810 INPUT CITY$
820 GOSUB 3000
830 RETURN
900 PRINT "ZIP: ";
910 INPUT ZIP$
920 GOSUB 3000
930 RETURN
1000 PRINT "BALANCE: ";
1010 INPUT BAL
1020 GOSUB 3000
1030 RETURN
2000 REM CHECK RECORD NOT PRESENT ERROR
2010 INPUT#15,ERRNUM,ERRNUM$
2020 IF ERRNUM=50 OR ERRNUM=0 THEN RETURN
2030 PRINT ERRNUM,ERRNUM$
2040 CLOSE 2
2050 CLOSE 15
2060 END
2070 RETURN
3000 REM WRITE RECORD TO FILE
3010 PRINT#15,"P"CHR$(2)CHR$(J)CHR$(0)CHR$(1)
3020 GOSUB 2000
3030 PRINT#2,FIRST$;",";LAST$;",";STREET$;
",";CITY$;",";ZIP$;",";BAL
3040 RETURN
4000 PRINT "DO YOU WISH TO MODIFY"
4010 PRINT "ANOTHER RECORD? (Y/N) ";
4020 INPUT ANS$
4030 IF ANS$="Y" THEN 150
4040 GOTO 430
4050 RETURN

```

Save this program as MODIFYRECORDS.

### **Example 6 - Menu-Driven Mailing Program**

ADDRECORDS, LABELS, SELECTEDLABELS, and MODIFY-RECORDS can be gathered together as subprograms in a large program. You choose a task for the computer to perform from a menu or list of available tasks displayed on the screen by the first few lines of the program. You can easily use this menu-driven program to maintain a mailing list for billing.



The program should begin by printing the activities from which you can select. The program might print

1. ADD A NEW RECORD
2. COMPLETE SET OF LABELS
3. SELECTED SET OF LABELS
4. MODIFY A RECORD
5. EXIT THE PROGRAM

The beginning of the program follows.

```

100 PRINT "0.  ENTER RECORDS FOR THE
FIRST TIME"
110 PRINT "1.  ADD A NEW RECORD"
120 PRINT "2.  COMPLETE SET OF LABELS"
130 PRINT "3.  SELECTED SET OF LABELS"
140 PRINT "4.  MODIFY A RECORD"
150 PRINT "5.  EXIT THE PROGRAM"
160 PRINT "CHOOSE A NUMBER ";
170 INPUT N
180 IF N=5 THEN END
190 IF N=0 THEN 10000
200 ON N GOTO 11000,12000,13000,14000

```

Each of the five subprograms should be renumbered as it is gathered into the large program and a REMARK statement should be added at the beginning of each subprogram.

To make the computer return to the main menu above after it has executed one of the subprograms at 10000, 11000, 12000, 13000 or 14000, you need to replace the END statement in each of the subprograms ENTERDATA, ADDRECORDS, LABELS, SELECTEDLABELS, and MODIFYRECORDS. For example, the ADDRECORDS subprogram would begin with

```

11000 REM ADDRECORDS SUBPROGRAM
11110 OPEN 15,8,15

```

The END statement in line 340 should be modified as follows:

```

11340 GOTO 110

```

You go to 110 since the ENTERDATA subprogram should be run only once.

**Example 7 - Precision Record Access**

In the previous examples, you used the PRINT# and INPUT# commands to write to and read from a record in a file using commas as item separators. You did not concern yourself with the precise location of each item in a record. In this example, you will place each data item (or field) of a record in a specified position in the record. You will use the last CHR\$ function in the "P" PRINT# statement to do this. Additional checking of the length of strings will be necessary to avoid one field overrunning the next field.

Here you will use a record with three fields: two string fields and one numeric field. The fields are read back from the record on diskette starting at the second position in each field. The complete program follows.

```

100 OPEN 15,8,15
110 OPEN 2,8,2,"EXAMPLE,L," + CHR$(40)
120 READ DES$,PRICE,COLR$
130 DATA SWIMSUIT,27.50,BLUE
140 REM PAD STRING VARIABLES WITH SPACES
150 LET SPACES$=" "
160 LET DES$=MID$(DES$,1,14)
170 LET PAD=14 - LEN(DES$)
180 LET DES$=DES$ + MID$(SPACES$,1,PAD)
190 LET COLR$=MID$(COLR$,1,7)
200 LET PAD=7 - LEN(COLR$)
210 LET COLR$=COLR$+MID$(SPACES$,1,PAD)
220 REM WRITE IN POSITIONS 1,16,32
230 LET P=1
240 GOSUB 2000
250 PRINT#2,DES$
260 LET P=16
270 GOSUB 2000
280 PRINT#2,PRICE
290 LET P=32
300 GOSUB 2000
310 PRINT#2,COLR$
320 PRINT "VALUES WRITTEN TO FILE"
330 PRINT DES$,PRICE,COLR$

```

```

340 REM READ VALUES FROM FILE TO FDES$,
FPRICE,FCOLR$ (BEGIN AT 2ND POSITION)
350 LET P=2
360 GOSUB 2000
370 INPUT#2,FDES$
380 LET P=17
390 GOSUB 2000
400 INPUT#2,FPRICE
410 LET P=33
420 GOSUB 2000
430 INPUT#2,FCOLR$
440 PRINT
450 PRINT "VALUES READ FROM FILE"
460 PRINT FDES$,FPRICE,FCOLR$
470 CLOSE 2
480 CLOSE 15
490 END
2000 REM LOCATE POSITION P IN RECORD 5
SUBROUTINE
2010 PRINT#15,"P"CHR$(2)CHR$(5)CHR$(0)
CHR$(P)
2020 RETURN

```

## 12-5 PROBLEMS

1. Design an appropriate record structure for a relative file that will be used to index your cassette tape collection. Be sure to indicate the maximum length of each item in the record and the total length in characters of the record. Determine valid Commodore 64 BASIC variable names for each item in the record structure.
2. Design an appropriate record structure for a random access file that will be used to keep track of your checkbook entries.
3. Design an appropriate record structure for a random access file that will be used to inventory your household possessions.
4. Write out an appropriate record structure for a file that will be used to keep your mailing list. Remember that birthdays and anniversaries are important to your friends.

5. Write a program that will use a file called CHARGE to manage your charge cards. Each record should have the following structure.

Variable	Description	Approximate Length
CARD\$	Name of Card	20
NAME\$	Name of Store	30
DATE\$	Date of Purchase	10
DES\$	Description of Purchase	50
AMT	Amount of Purchase	8

The program should allow you to total the amount of money you have charged to each card in the entire file.

6. Write a program that uses the record structure from problem 4 to manage your personal mailing list. The program should allow you to print labels for your Christmas cards and for messages to your friends at work.

## 12-6 PRACTICE TEST

1. If the following program is executed

```

90 OPEN 15,8,15
100 OPEN 2,8,2,"FILETWO,L," + CHR$(70)
110 PRINT#15,"P"CHR$(2)CHR$(5)CHR$(0)CHR$(1)
120 PRINT#2,"HELLO";",","GEORGE"
130 CLOSE 2
140 CLOSE 15
150 END

```

- a. What file will be used?

---

- b. How many characters are allowed in each record?

---

- c. How many items are placed in the record?

---

d. Which record will be written to?

---

2. Write a program line that opens a relative file named INVENTORY with record length 45.
- 

3. Write a program that will read five information items from record 75 in a relative file named TEST1 with record length 50. Be sure to close the file.
- 

4. What is wrong with the following program line?

```
200 OPEN 2,8,2,"FILEONE,L", CHR$(70)
```

---

5. What is wrong with the following program lines for reading **FIRST\$** from record 5 of a data file named FILEONE?

```
200 OPEN 15,8,15
210 OPEN 2,8,2,"FILEONE,L," + 100
220 PRINT# 15,CHR$(5)CHR$(2)CHR$(0)CHR$(1)
230 INPUT FIRST$
```

---



# OTHER CONFIGURATIONS

## **Commodore 64 without a Disk Drive**

This book can be used effectively without a disk drive. Chapter 12 on files, however, must be omitted along with the sections of Chapter 3 on program management.

## **Commodore 64 with a Commodore Datacassette Recorder**

In Chapter 3, you may use a Datacassette recorder instead of a disk drive to store and retrieve programs. You must refer to the *Commodore 64 User's Guide* for instructions on how to use the Datacassette recorder to do this.

The chapter on files requires disk drive storage and retrieval and must be omitted if you are using the Datacassette recorder.





# SUPPORT PROGRAM

## **The C-64 Wedge or DOS Support Program**

The C-64 Wedge or DOS Support Program on the Demonstration (Test/Demo) Disk allows you to control the disk drive more easily. You may wish to save this program and its companion program DOS 5.1 on each of the diskettes you use. To use this program, you must type

```
LOAD "C-64 WEDGE",8  
RUN
```

each time you turn on the computer. Once the program is loaded and run, you may use / for loading programs without using quotes or the comma and the 8. For example,

```
/AVERAGE
```

will load the program AVERAGE into memory.

The @ key is now used to send commands to the disk drive. For example,

```
@S
```

displays the directory and files on the screen without erasing the program in memory.

@ is now used to format diskettes and to erase programs and files from diskette. For example,

```
@NEW@ : BOOKS, ZZ
```

formats a diskette as BOOKS with id ZZ, and

`@SCRATCH0:HELLO`

erases the program HELLO from the diskette in the disk drive.

Finally, disk drive errors signalled by a blinking red light on the disk drive can be read from the disk drive and displayed on the screen simply by typing

`@` (and `RETURN` )

Further information on the C-64 Wedge program is available in Chapter 3 of the *VIC-1541 Single Drive Floppy Disk User's Manual*.

# GLOSSARY

**ABS(X)** A BASIC function that takes the absolute value of X. Positive values of X remain positive. Negative values of X become positive.

**Arithmetic Operators** Addition +, subtraction −, multiplication \*, division /, and exponentiation ↑.

**ASC(A\$)** A BASIC function that converts the first character in A\$ to its equivalent position number in the ASCII character set.

**BASIC** An acronym for "Beginners All-Purpose Instruction Code". More people know how to program computers in BASIC than in any other language.

**CHR\$(N)** A BASIC function that returns the Nth character from the ASCII character set.

**CLOSE** A statement that closes an information path.

**Cursor keys** Keys at the right of the keyboard used to move the cursor around the screen. Useful in editing program lines.

**Commodore key** A key marked **C=** on the far left side of the keyboard used to access graphics characters and features.

**DATA** A statement used to hold information within a program. This information is called for with the READ statement.

**DEF** A statement used to define lengthy functions that will be used often in a program.

**Deleting BASIC Statements** Type the line number of the statement to be deleted and then press **RETURN** .

**DIM** A statement used to specify the size of, and reserve space for, arrays.

**Double Subscripts** Indicated within parentheses following a variable name, and separated by a comma. Used to specify a row and column number in an array. A(3,5), for example, means the element in the two dimensional array A at row 3 and column 5.

**E Notation** A notation used in BASIC to express either very large or very small numbers.

**END** Marks the end of a BASIC program or the end of the main program.

**File** A collection of information.

**FOR NEXT** Statements used in BASIC to set up loops.

**GET** A statement that assigns the next key pressed to a variable.

**GOSUB** A statement used to transfer program control to a sub-routine.

**GOTO** An unconditional branch statement.

**Graphics characters** Special characters that appear on the fronts of some of the keys on the Commodore 64. These characters are accessed with the SHIFT and Commodore (C=) keys when the computer is in upper case/graphics mode.

**IF THEN** A conditional branch statement.

**INPUT** A statement that calls for input of information from the keyboard or the disk drive.

**Inserting BASIC Statements** Type in the statement using a line number not already in use.

**INT(X)** A BASIC function that takes the integer part of X. The integer part of X is defined as the first integer less than or equal to X.

**LEN(A\$)** A BASIC function used to determine the length of a string in characters. For example, if A\$ = "HOUSE" then LEN(A\$) is 5.

**LET** Identifies an assignment statement. It is always followed by a variable name, an equal sign, and a BASIC expression. The LET in the assignment statement is optional.

**LIST** A command used to tell the computer to print out the program in memory. May be used in editing program lines.

**LOAD** A command that loads a copy of a program on a diskette into the computer memory.

**MID\$** A statement that returns the requested part of a given string. For example, if `A$ = "TOGETHER"`, then `MID$(A$,5,3)` will return the string "THE".

**NEW** A command that erases the current program in memory.

**Numeric Variable Names** The Commodore 64 computer allows variable names up to 79 characters long. The first character must be a letter.

**OPEN** A statement that assigns an information path.

**PRINT** A statement that sends information from the computer to the screen or the disk drive.

**Random Numbers** A sequence of numbers generated by the `RND` function. They appear to have no pattern or relationship to one another.

**READ** A statement that calls for input of information stored in `DATA` statements within the program.

**Record** A structured set of information.

**Replacing BASIC Statements** Retype the statement to be replaced including the line number.

**RETURN** A statement used to transfer program control back from a subroutine to the main program.

**RND** A BASIC function used to generate random numbers.

**RUN** A command used to tell the computer to begin execution of the program in memory.

**SAVE** A command that saves a program from memory to diskette. For example, `SAVE "AVERAGE",8` would save the program currently in memory to the diskette in the disk drive under the name `AVERAGE`.

**SGN(X)** A BASIC function that determines the sign of `X`. `SGN(X)` is +1, 0, -1 as `X` is positive, zero, or negative respectively.

**Single Subscripts** Indicated within parentheses following a variable name. Used to specify a particular element in an array. `A(6)`, for example, means the sixth element of the one dimensional array `A`.

**SQR(X)** A BASIC function that takes the square root of `X`. `X` cannot be negative.

**String Variable Names** Commodore 64 BASIC allows string variable names of up to 79 characters long. They must start with a letter and end with a \$ sign.

**TAB** A BASIC function that will tab to a specific position when in a PRINT statement.

**Upper case/graphics mode** The start-up screen mode in which pressing a letter key places a capital letter on the screen.

**Upper/lower case mode** A screen mode accessed by pressing the Commodore key (C=) while holding down the SHIFT key. In this mode, capital letters are obtained by using the SHIFT key.

# PRACTICE TEST SOLUTIONS

## Chapter 1

1. Press the RETURN key.
2. Hold down `RUN/STOP` and press `RESTORE`. Alternately, you may turn the machine off and back on again.
3. \*
4. Hold down `SHIFT` and press `CLR/HOME`.
5. Division.
6. The number 2 will be displayed on the screen.
7. The string  $25/5+2$  will be displayed on the screen.
8. Move the cursor to the left with the delete key (on the right side of the keyboard) to the G in PRING. Then type  $T\ 2+3*4$ .

## Chapter 2

1. Press the RETURN key.
2. Press `RUN/STOP|RESTORE`.
3. Press `RUN/STOP`.
4. The statement `PRINT C` has no line number.

5. The number 2 would be displayed on the screen.
6. Up to 79 characters.
7. Type the line using a line number not already in the program.
8. Retype the line including the line number.
9. Type the line number and press the RETURN key.
10. Type LIST and press `RETURN`.
11. Press `SHIFT|CLR/HOME`.
12. Type RUN and press `RETURN`.
13. Type NEW and press `RETURN`.
14. A character string variable always ends with a \$.
15. Displays line 120 for editing.
16. The cursor keys `←CRSR⇒` and `↑CRSR↓` and sometimes the SHIFT key (for `←` and `↑`).
17. `SHIFT|INST/DEL` and `INST/DEL`.
18. By pressing `RETURN`.

### Chapter 3

1. a. \*   b. ↑   c. /
2. a. Exponentiation   b. Multiplication and division   c. Addition and subtraction
3. Addition.
4. Left to right.



5. 100 LET A=(4+3\*B/D)↑2
6. 4
7. a. 5.16E+09 b. 3.14E-05
8. a. 7258000 b. 0.001437
9. / then + then ↑
10. a. Type LOAD"(name of program)",8 and press RETURN.  
 b. Type SAVE"(name of program)",8 and press RETURN.  
 c. OPEN 15,8,15,"SCRATCH0:(name of program)".  
 d. Type NEW and press RETURN.  
 e. Type LIST and press RETURN.  
 f. Type RUN and press RETURN.  
 g. Type LOAD "\$",8 RETURN.  
 LIST RETURN.
11. Use the cursor keys on the right side of the keyboard to move the cursor to the error. Type the correct character and press RETURN to enter the line in the program.

#### Chapter 4

1. 1 2 3 4 5 6 7 8 9 10 11 12  
 13 14 15 16 17 18 19 20 21 22 23 24  
 etc.
2. a. By assignment (e.g., 100 LET A=3) b. INPUT statements  
 c. READ and DATA statements
3. A string.
4. To provide information within the program for the benefit of the programmer or user.
5. DATA.
6. Y = 3 will be printed out.

7. Four.
8. As many as needed.
9. To provide a method to obtain variable spacing in the output.
10. 

```
1      3
1 3
```
11. 

```
?10,12,13
?EXTRA IGNORED
22
READY.
```
12. 

```
100 PRINT "INPUT NO. OF MILES";
110 INPUT N
120 LET K=1.609*N
130 PRINT N;" MILES EQUAL ";K;"KILOMETERS"
140 END
```

## Chapter 5

1. 

```
6
10
14
18
```
2. BEST  
  
BETTER  
BEST  
  
GOOD  
BETTER  
BEST  
  
OUT OF DATA ERROR IN LINE 100
3. 

```
100 PRINT "HOW MANY WIDGETS";
110 INPUT N
120 IF N <= 20 THEN 160
```

```
130 IF N <= 50 THEN 180
140 LET U=1.5
150 GOTO 190
160 LET U=2
170 GOTO 190
180 LET U=1.8
190 LET P=N*U
200 PRINT "PRICE PER WIDGET IS ";U
210 PRINT "TOTAL COST OF ORDER IS";P
220 PRINT
230 GOTO 100
240 END
```

4. 

```
100 LET X=0
110 PRINT X;
120 LET X=X+5
130 IF X <= 175 THEN 110
140 END
```
5. 

```
100 PRINT "WHAT WAS SPEED LIMIT ";
110 INPUT A
120 PRINT "SPEED ARRESTED AT ";
130 INPUT B
140 LET X=B-A
150 IF X <= 10 THEN 210
160 IF X <= 20 THEN 230
170 IF X <= 30 THEN 250
180 IF X <= 40 THEN 270
190 LET F=80
200 GOTO 280
210 LET F=5
220 GOTO 280
230 LET F=10
240 GOTO 280
250 LET F=20
260 GOTO 280
270 LET F=40
280 PRINT "FINE IS ";F;" DOLLARS"
290 END
```

**Chapter 6**

1. 20 18 16 14 12 10 8 6 4 2
2. 1 2 3 2 4 6 3 6 9 4 8 12
3. a. 6 b. 7 c. 22.8 d. -1
4. The loops are crossed.
5.
 

```

100 PRINT "MILES","KILOMETERS"
110 PRINT "-----","-----"
120 PRINT
130 FOR M=10 TO 100 STEP 10
140 PRINT M,1.609*M
150 NEXT M
160 END
      
```
6.
 

```

100 DATA 10
110 DATA 25,21,24,21,26,27,25,24,23,24
120 READ N
130 LET S=0
140 FOR I=1 TO N
150 READ X
160 LET S=S+X
170 NEXT I
180 PRINT S/N
190 END
      
```

**Chapter 7**

1. To save space for an array.
2. X(3,4)
3.
 

```

100 DIM A(50)
110 PRINT "HOW MANY NUMBERS ";
120 INPUT N
130 PRINT "WHAT ARE THE NUMBERS"
140 FOR I=1 TO N
150 INPUT A(I)
160 NEXT I
      
```

```

170 LET S=0
180 FOR I=1 TO N
190 IF A(I) <= 0 THEN 210
200 LET S=S+A(I)
210 NEXT I
220 PRINT "SUM OF POSITIVE ELEMENTS IS ";S
230 END

```

4. 

```

100 FOR R=1 TO 4
110 FOR C=1 TO 6
120 LET X(R,C)=4
130 NEXT C
140 NEXT R
150 END

```

5. 

```

2 0 0 0 0
0 2 0 0 0
0 0 2 0 0
0 0 0 2 0
0 0 0 0 2

```

6. a.  $100 \text{ DIM } A(2,3)$    b.  $A(2,3) = 4$    c.  $A(X,Y) = A(1,2) = 3$   
d.  $A(A(1,1), A(2,2)) = A(1,2) = 3$

## Chapter 8

1. By appending \$ to a numeric variable name.
2. False
3. `MID$(A$,5,9)`
4. 

```

100 INPUT A$
110 FOR X=LEN(A$) TO 1 STEP -1
120 PRINT MID$(A$,1,X)
130 NEXT X
140 END

```

5. A  
AB  
ABC  
ABCD  
ABCDE

(etc.)

ABCDEFGHIJKLMNOPQRSTUVWXYZ

### Chapter 9

1. a. 4 b. 14 c. 30 d. 80
2. 2 1 3  
4
3. a. Type in GOSUB (line number at beginning of subroutine)  
b. RETURN
4. WHITE  
RED  
BLUE

### Chapter 10

1. 6
2. The screen will be cleared and the cursor will be placed in HOME position.
3. A reverse J will be printed.
4. A red square will be printed.
5. Hold down the Commodore key and press the Q key.

### Chapter 11

1. 100 FOR I=1 TO 100  
110 LET X=INT(4\*RND(1))+1)  
120 PRINT X;  
130 NEXT I  
140 END

```

2. 100 FOR I=1 TO 100
    110 LET X=25+25*RND(1)
    120 PRINT X,
    130 NEXT I
    140 END

```

3. The output will be randomly selected from WHITE and RED. Three program outputs are shown to indicate the random nature of the process.

(1)	(2)	(3)
RED	WHITE	WHITE
RED	WHITE	RED
WHITE	RED	WHITE
WHITE	WHITE	WHITE
RED	WHITE	WHITE
RED	RED	WHITE
RED	RED	RED
WHITE	RED	WHITE
RED	WHITE	WHITE
RED	WHITE	RED

4. Five random numbers of the form X.XX over the range 0.00 to 9.99. Three program outputs are shown below to illustrate the random nature of the process.

(1)	(2)	(3)
0.51	6.69	1.15
9.34	4.04	8.87
9.08	9.06	9.26
9.26	6.71	2.59
5.98	8.15	3.05

## Chapter 12

- a. FILETWO    b. 70    c. 2    d. 5
- 100 OPEN 2,8,2,"INVENTORY,L" + CHR\$(45)

3. 90 OPEN 15,8,15  
100 OPEN 2,8,2,"TEST1,L" + CHR\$(50)  
110 PRINT#15,"P"CHR\$(2)CHR\$(75)CHR\$(0)CHR\$(1)  
120 INPUT#2 A\$,B\$,C\$,D\$,E\$  
130 CLOSE 2  
140 CLOSE 15  
150 END

4. The correct program line is:

200 OPEN 2,8,2,"FILEONE,L" + CHR\$(70)

5. Lines 210 and 220 should be:

210 OPEN 2,8,2,"FILEONE,L," + CHR\$(100)  
220 PRINT#15,"P"CHR\$(2)CHR\$(5)CHR\$(0)CHR\$(1)



# SOLUTIONS TO ODD-NUMBERED PROBLEMS

## Chapter 4

1. 

```
100 REM CHAP 4, PROB 1
110 READ A,B,C,D
120 DATA 10,9,1,2
130 LET S=A+B
140 LET P=C*D
150 PRINT S,P
160 END
```
3. 

```
100 REM CHAP 4, PROB 3
110 READ A,B,C,D
120 DATA 21,18,6,3
130 PRINT A
140 PRINT B
150 PRINT C
160 PRINT D
170 END
```
5. There is no value assigned to C.
7. 

```
100 REM CHAP 4, PROB 7
110 PRINT "CASH = ";
120 INPUT C
130 PRINT "MARKETABLE SECURITIES = ";
140 INPUT M
150 PRINT "RECEIVABLES = ";
160 INPUT R
170 PRINT "LIABILITIES = ";
180 INPUT L
190 LET A=(C+M+R)/L
200 PRINT "ACID-TEST RATIO = ";A
210 END
```

9. The program loops back to line 100 where A is set equal to 1 after each printout. The program can be corrected by changing line 130 as follows.

```
130 GOTO 110
```

11. The problem lies in statements 100, 110, and 120. The values of L, W, and H are supposed to be printed out, but they haven't been defined. The computer will assign the value zero to the three variables and these zeros are printed out by lines 100, 110, 120. The program may be corrected by deleting line 130 and L, W, and H at the ends of lines 100, 110, and 120 and adding:

```
105 INPUT L
115 INPUT W
125 INPUT H
```

13. 100 REM CHAP 4, PROB 13  
 110 DATA 21423,21493,5  
 120 DATA 5270,5504,13  
 130 DATA 65214,65559,11.5  
 140 READ R1,R2,G  
 150 LET M=(R2-R1)/G  
 160 PRINT M  
 170 GOTO 140  
 180 END

15. 100 REM CHAP 4, PROB 15  
 110 DATA 92,63,75,82,72,53,100,89,70,81  
 120 READ A,B,C,D,E,F,G,H,I,J  
 130 PRINT (A+B+C+D+E+F+G+H+I+J)/10  
 140 END

17. 100 REM CHAP 4, PROB 17  
 110 PRINT "QUOTED INTEREST RATE (PERCENT) "  
 120 INPUT R  
 130 PRINT "NO. OF TIMES COMPOUNDED PER YEAR "  
 140 INPUT M  
 150 LET T=((1+R/(100\*M))<sup>M</sup>-1)\*100  
 160 PRINT "TRUE ANNUAL INTEREST RATE IS"  
 170 PRINT T  
 180 END

```
19. 100 REM CHAP 4, PROB 19
    110 PRINT "INITIAL INVESTMENT ";
    120 INPUT P
    130 PRINT "ANNUAL INTEREST RATE(%) ";
    140 INPUT I
    150 PRINT "YEARS LEFT TO ACCRUE INTEREST ";
    160 INPUT N
    170 LET T=P*(1+I/100)^N
    180 PRINT "TOTAL VALUE IS ";T
    190 END
```

### Chapter 5

- ```
1. 100 REM CHAP 5, PROB 1
    110 INPUT X,Y
    120 IF X>Y THEN 150
    130 PRINT Y
    140 GOTO 160
    150 PRINT X
    160 END

3. 100 REM CHAP 5, PROB 3
    110 LET S=0
    120 LET X=1
    130 LET S=S+X
    140 LET X=X+1
    150 IF X <= 100 THEN 130
    160 PRINT S
    170 END

5. OUT OF DATA ERROR IN LINE 120

7. 100 REM CHAP 5, PROB 7
    110 LET S=0
    120 READ X
    130 IF X=9999 THEN 180
    140 IF X<-10 THEN 120
    150 IF X>10 THEN 120
    160 LET S=S+X
    170 GOTO 120
    180 PRINT S
    190 DATA -1,22,17,-6,4,7,9999
    200 END
```

9. 100 REM CHAP 5, PROB 9  
110 LET C=1  
120 LET T=0  
130 LET W=1  
140 LET T=T+W  
150 LET C=C+1  
160 LET W=2\*W  
170 IF C <= 22 THEN 140  
180 PRINT T/100  
190 END
11. The number 83 will be printed out. The program finds the largest number contained in the two DATA statements.
13. 100 REM CHAP 5, PROB 13  
110 PRINT "LIST PRICE (\$) ";  
120 INPUT L  
130 PRINT "DISCOUNT RATE (%) ";  
140 INPUT R  
150 LET D=L\*(1 - R/100)  
160 PRINT "DISCOUNTED PRICE IS"  
170 PRINT D;" DOLLARS"  
180 END
15. 100 REM CHAP 5, PROB 15  
110 INPUT A,B  
120 IF A >= 10 THEN 140  
130 GOTO 180  
140 IF B >= 10 THEN 160  
150 GOTO 180  
160 PRINT A+B  
170 GOTO 280  
180 IF A < 10 THEN 200  
190 GOTO 240  
200 IF B < 10 THEN 220  
210 GOTO 240  
220 PRINT A\*B  
230 GOTO 280  
240 IF A < B THEN 270  
250 PRINT A-B  
260 GOTO 280  
270 PRINT B-A  
280 END

```
17. 100 REM CHAP 5, PROB 17
110 PRINT "GROWTH RATE (%) ";
120 INPUT R
130 LET N=0
140 LET Q=1
150 LET Q=Q*(1+R/100)
160 LET N=N+1
170 IF Q <= 2 THEN 150
180 PRINT "NUMBER OF GROWTH PERIODS TO DOUBLE
IS ";N
190 END
```

## Chapter 6

```
1. 100 REM CHAP 6, PROB 1
    110 PRINT "N","SQ(R(N))"
    120 PRINT
    130 FOR N=2 TO 4 STEP .2
    140 PRINT N,SQ(R(N))
    150 NEXT N
    160 END
```

```
3. 100 REM CHAP 6, PROB 3
    110 INPUT N
    120 FOR X=2 TO N STEP 2
    130 PRINT X
    140 NEXT X
    150 END
```

```

5.      ABCDEFGHIJ
        ABCDEFGHIJ
          ABCDEFGHIJ
            ABCDEFGHIJ
              ABCDEFGHIJ
                ABCDEFGHIJ
                  ABCDEFGHIJ
                    ABCDEFGHIJ
                      ABCDEFGHIJ
                        ABCDEFGHIJ

```

7. Since the Z and V loops are crossed, a NEXT WITHOUT FOR ERROR message will appear.
9. It reads and prints out five numbers rounded off to two places past the decimal point.
11. Prints out 1.
13.
 

```

100 REM CHAP 6, PROB 13
110 INPUT N
120 INPUT X
130 LET L=X
140 LET H=X
150 LET S=X
160 FOR I=1 TO N-1
170 INPUT X
180 IF X > L THEN 200
190 LET L=X
200 IF X < H THEN 220
210 LET H=X
220 LET S=S+X
230 NEXT I
240 PRINT "HIGHEST GRADE IS ";H
250 PRINT "LOWEST GRADE IS ";L
260 PRINT "AVERAGE IS ";S/N
270 END
      
```
15.
 

```

1 2 3
2 4 6
3 6 9
4 8 12
      
```
17.
 

```

100 REM CHAP 6, PROB 17
110 READ N
120 FOR I=1 TO N
130 READ M,R,D1,D2,D3,D4,D5
140 PRINT "EMPLOYEE NUMBER ";M
150 LET H=D1+D2+D3+D4+D5
160 IF H <= 40 THEN 190
170 LET P=R*40 + 1.5*R*(H-40)
180 GOTO 200
190 LET P=R*H
200 PRINT "PAY IS ";P
210 NEXT I
      
```

```
220 DATA 5
230 DATA 2,4,8,8,10,8,7,10
240 DATA 5,3.75,7,8,8,6,10
250 DATA 1,3.25,8,10,6,8,8
260 DATA 4,5,8,10,6,10,6
270 DATA 3,4.25,6,6,8,10,7
280 END
```

### Chapter 7

1. 

```
100 REM CHAP 7, PROB 1
110 DIM X(20)
120 READ N
130 FOR I=1 TO N
140 READ X(I)
150 NEXT I
160 FOR I=1 TO N
170 PRINT X(I)
180 NEXT I
190 DATA 12
200 DATA 2,1,4,3,2,4,5,6,3,5,4,1
210 END
```
3. 

```
100 REM CHAP 7, PROB 3
110 DIM A(10,10)
120 INPUT N
130 FOR R=1 TO N
140 FOR C=1 TO N
150 INPUT A(R,C)
160 NEXT C
170 NEXT R
180 LET S=0
190 FOR I=1 TO N
200 LET S=S + A(I,I)
210 NEXT I
220 PRINT "SUM OF MAIN DIAGONAL IS ";S
230 END
```
5. 

```
100 REM CHAP 7, PROB 5
110 DIM A(15,15)
120 INPUT M,N
```

```

130 FOR R=1 TO M
140 FOR C=1 TO N
150 INPUT A(R,C)
160 NEXT C
170 NEXT R
180 LET S=0
190 FOR R=1 TO M
200 FOR C=1 TO N
210 LET S=S+A(R,C)
220 NEXT C
230 NEXT R
240 PRINT "SUM OF ENTRIES IS ";S
250 END

```

7. 10

9. 16

```

11. 100 REM CHAP 7, PROB 11
    110 DIM X(100)
    120 INPUT N
    130 FOR I=1 TO N
    140 INPUT X(I)
    150 NEXT I
    160 FOR I=1 TO N-1
    170 IF X(I) >= X(I+1) THEN 220
    180 LET T=X(I+1)
    190 LET X(I+1)=X(I)
    200 LET X(I)=T
    210 GOTO 160
    220 NEXT I
    230 FOR I=1 TO N
    240 PRINT X(I);
    250 NEXT I
    260 END

```

```

13. 1 1 1 1 1 1
    0 0 0 0 0 0
    0 0 1 1 1 1
    0 0 0 0 0 0
    0 0 0 0 1 1
    0 0 0 0 0 0

```



15. 100 REM CHAP 7, PROB 15

```
110 DIM X(2,5)
120 FOR R=1 TO 2
130 FOR C=1 TO 5
140 READ X(R,C)
150 NEXT C
160 NEXT R
170 DATA 2,1,0,5,1
180 DATA 3,2,1,3,1
190 FOR R=1 TO 2
200 FOR C=1 TO 5
210 PRINT X(R,C);
220 NEXT C
230 PRINT
240 NEXT R
250 END
```

17. 100 REM CHAP 7, PROB 17

```
110 DIM X(20,20)
120 INPUT M,N
130 FOR R=1 TO M
140 FOR C=1 TO N
150 INPUT X(R,C)
160 NEXT C
170 NEXT R
180 FOR R=1 TO M
190 LET S=0
200 FOR C=1 TO N
210 LET S=S+X(R,C)
220 NEXT C
230 PRINT "SUM OF ROW ";R;" IS ";S
240 NEXT R
250 FOR C=1 TO N
260 LET P=1
270 FOR R=1 TO M
280 LET P=P*X(R,C)
290 NEXT R
300 PRINT "PRODUCT OF COLUMN ";C;" IS ";P
310 NEXT C
320 END
```

- ```
19. 100 REM CHAP 7, PROB 19
    110 DIM X(4,6)
    120 FOR R=1 TO 4
    130 FOR C=1 TO 6
    140 READ X(R,C)
    150 NEXT C
    160 NEXT R
    170 DATA 48,40,73,120,100,90
    180 DATA 75,130,90,140,110,85
    190 DATA 50,72,140,125,106,92
    200 DATA 108,75,92,152,91,87
    210 FOR C=1 TO 6
    220 LET S=0
    230 FOR R=1 TO 4
    240 LET S=S+X(R,C)
    250 NEXT R
    260 PRINT "TOTAL-DAY ";C;" IS ";S
    270 NEXT C
    280 PRINT
    290 FOR R=1 TO 4
    300 LET S=0
    310 FOR C=1 TO 6
    320 LET S=S+X(R,C)
    330 NEXT C
    340 PRINT "TOTAL-SALESPERSON ";R;" IS ";S
    350 NEXT R
    360 LET S=0
    370 FOR R=1 TO 4
    380 FOR C=1 TO 6
    390 LET S=S+X(R,C)
    400 NEXT C
    410 NEXT R
    420 PRINT
    430 PRINT "TOTAL SALES FOR THE WEEK IS";S
    440 END

21. 100 REM CHAP 7, PROB 21
    110 DIM P(20),X(20)
    120 PRINT "HOW LONG ARE THE ARRAYS ";
    130 INPUT N
```

```
140 FOR I=1 TO N
150 LET P(I)=I
160 NEXT I
170 FOR R=1 TO N
180 INPUT X(R)
190 NEXT R
200 FOR I=1 TO N-1
210 IF X(P(I))>X(P(I+1)) THEN 260
220 LET T=P(I)
230 LET P(I)=P(I+1)
240 LET P(I+1)=T
250 GOTO 200
260 NEXT I
270 PRINT " P", " X"
280 PRINT
290 FOR I=1 TO N
300 PRINT P(I),X(I),X(P(I))
310 NEXT I
320 END
```

### Chapter 8

1. 

```
100 REM CHAP 8, PROB 1
110 INPUT A$
120 FOR I=1 TO LEN(A$)
130 PRINT MID$(A$,I,1)
140 NEXT I
150 END
```
3. 

```
100 REM CHAP 8, PROB 3
110 INPUT A$
120 LET A=0
130 LET E=0
140 LET I=0
150 LET O=0
160 LET U=0
170 FOR J=1 TO LEN(A$)
180 IF MID$(A$,J,1) = "A" THEN 240
190 IF MID$(A$,J,1) = "E" THEN 260
200 IF MID$(A$,J,1) = "I" THEN 280
210 IF MID$(A$,J,1) = "O" THEN 300
220 IF MID$(A$,J,1) = "U" THEN 320
```

```
230 GOTO 330
240 LET A=A+1
250 GOTO 330
260 LET E=E+1
270 GOTO 330
280 LET I=I+1
290 GOTO 330
300 LET O=O+1
310 GOTO 330
320 LET U=U+1
330 NEXT J
340 PRINT "A = ";A
350 PRINT "E = ";E
360 PRINT "I = ";I
370 PRINT "O = ";O
380 PRINT "U = ";U
390 END
```

5. 100 REM CHAP 8, PROB 5  
110 INPUT A\$  
120 FOR I=1 TO LEN(A\$)  
130 IF MID\$(A\$,I,1)=CHR\$(32) THEN 150  
140 LET B\$=B\$ + MID\$(A\$,I,1)  
150 NEXT I  
160 PRINT B\$  
170 END

7. 100 REM CHAP 8, PROB 7  
110 LET C=0  
120 FOR K=1 TO 5  
130 INPUT A\$  
140 IF MID\$(A\$,1,4)<>"THE " THEN 160  
150 LET C=C+1  
160 FOR I=2 TO LEN(A\$)-3  
170 IF MID\$(A\$,I,4)<>"THE " THEN 190  
180 LET C=C+1  
190 NEXT I  
200 NEXT K  
210 PRINT C  
220 END

```
9. 100 REM CHAP 8, PROB 9
    110 INPUT A$
    120 LET C=0
    130 FOR K=1 TO LEN(A$)-1
    140 IF MID$(A$,K,2)<>"IN" THEN 160
    150 LET C=C+1
    160 NEXT K
    170 PRINT C
    180 END
```

11. Delete lines 190 through 250. Add lines 185 and 355 as follows.

```
185 FOR R=1 TO 3
355 NEXT R
```

### Chapter 9

```
1. 25 5 20
    65
```

```
3. 100 REM CHAP 9, PROB 3
    110 DEF FNA(R)=3.14159*R↑2
    120 DEF FNB(R)=4*3.14159*R↑3/3
    130 PRINT
    140 PRINT " R","AREA OF","VOLUME OF"
    150 PRINT TAB(10);"CIRCLE","SPHERE"
    160 PRINT
    170 FOR R=1 TO 5 STEP .5
    180 PRINT R;TAB(7);FNA(R),FNB(R)
    190 NEXT R
    200 END
```

```
5. 55
    15
    36
```

```
7. 500 REM SUBROUTINE
    510 LET L=X(2)
    520 FOR I=3 TO X(1)+1
    530 IF L >= X(I) THEN 550
    540 LET L=X(I)
    550 NEXT I
    560 RETURN
```

9.
 

```

900 REM SUBROUTINE
910 LET S1=0
920 LET S2=0
930 FOR I=2 TO Y(1)+1
940 LET S1=S1+Y(I)
950 LET S2=S2+Y(I)2
960 NEXT I
970 LET M=S1/Y(1)
980 LET S=SQR((Y(1)*S2-S12)/(Y(1)*(Y(1)-1)))
990 RETURN
      
```

## Chapter 10

1. S OK will be printed.
3. Modify line 1010 as follows:
 

```

1010 IF R<1 OR R>25 OR C<1 OR C>40 THEN
PRINT "OFF THE SCREEN":END
      
```
5. A modification of Example 4 gives:
 

```

90 PRINT CHR$(28);
100 PRINT CHR$(147);
110 LET J=1
120 PRINT CHR$(19);
125 LET N=N + 1
130 FOR I=1 TO N
140 PRINT
150 NEXT I
160 PRINT TAB(J); "-----"
170 PRINT TAB(J); "|         |"
180 PRINT TAB(J); "|         |"
190 PRINT TAB(J); "-----"
200 LET J=J + 1
210 IF J > 33 THEN END
215 IF I > 20 THEN END
220 PRINT CHR$(147);
230 GOTO 120
240 END
      
```

**Chapter 11**

1. 

```
50 REM CHAP 11, PROB 1
100 FOR I=1 TO 25
110 LET N=INT(100*RND(1))/10
120 PRINT N,
130 NEXT I
140 END
```
3. A typical output is:  

```
.04 .02 .17 .14
.01 .03 .16 .04
.05 .17 .19 .12
.07 .11 .19 .11
.19 .2 .18 .04
```
5. 

```
50 REM CHAP 11, PROB 5
100 FOR I=1 TO 5
110 READ N
120 LET H=0
130 LET T=0
140 FOR J=1 TO N
150 LET X=INT(2*RND(1)+1)
160 IF X=1 THEN 190
170 LET T=T+1
180 GOTO 200
190 LET H=H+1
200 NEXT J
210 PRINT
220 PRINT "FOR ";N;" TOSSES THERE WERE"
230 PRINT H;" HEADS AND ";T;" TAILS"
240 NEXT I
250 DATA 10,50,100,500,1000
260 END
```
7. 

```
50 REM CHAP 11, PROB 7
100 LET S=0
110 FOR I=1 TO 100
120 LET S=S+RND(1)
130 NEXT I
140 PRINT S/100
150 END
```

9. 50 REM CHAP 11, PROB 9  
 100 LET M=0  
 110 FOR I=1 TO 1000  
 120 LET A=60\*RND(1)  
 130 LET B=60\*RND(1)  
 140 IF ABS(A-B)>10 THEN 160  
 150 LET M=M+1  
 160 NEXT I  
 170 PRINT "PROBABILITY OF A MEET IS ";M/1000  
 180 END
  
11. 50 REM CHAP 11, PROB 11  
 100 FOR I=1 TO 25  
 110 LET S=0  
 120 FOR J=1 TO 12  
 130 LET S=S+RND(1)  
 140 NEXT J  
 150 LET R=10+2\*(S-6)  
 160 PRINT INT(100\*R+.5)/100,  
 170 NEXT I  
 180 END
  
13. 50 REM CHAP 11, PROB 13  
 100 DIM A(11)  
 110 FOR N=1 TO 1000  
 120 LET SUM=INT(6\*RND(1)+1)+INT(6\*RND(1)+1)  
 130 FOR I=2 TO 12  
 140 IF SUM=I THEN LET A(I-1)=A(I-1) + 1:GOTO 160  
 150 NEXT I  
 160 NEXT N  
 170 FOR I=2 TO 12  
 180 PRINT I;A(I-1)  
 190 NEXT I  
 200 END

## Chapter 12

1. A possible record structure would be as follows.



Variable	Description	Approximate Length
CTITLE\$	Title of cassette	50
LABEL\$	Record company	20
NAME\$	Name of musicians	50
KIND\$	Category of music	15
PRICE	Price of cassette	5

The record length might be 45 where five extra spaces have been added for the item separation non-printing characters.

3. A possible record structure would be as follows.

Variable	Description	Approximate Length
NAME\$	Name of item	50
ROOM\$	Location of item	15
AMT	Value of item	10

The record length might be 78 where three spaces are added to separate the items.

5. A program for entering a month's charges and totalling them for a particular credit card follows.

```

100 REM CHAP 12, PROB 5
110 LET I=1
120 PRINT "NAME OF CARD: ";
130 INPUT CARD$
140 IF CARD$="QUIT" THEN 320
150 PRINT "NAME OF STORE: ";
160 INPUT NAMES$
170 PRINT "DATE OF PURCHASE: ";
180 INPUT DATES$
190 PRINT "DESCRIPTION OF PURCHASE: ";
200 INPUT DESS$
210 PRINT "COST OF PURCHASE: ";
220 INPUT AMT
230 OPEN 15,8,15

```

```

240 OPEN 2,8,2 "CHARGE,L," + CHR$(125)
250 PRINT#15,"P"CHR$(2)CHR$(I)CHR$(0)CHR$(1)
260 GOSUB 1000
270 PRINT#2,CARD$,",";NAME$,",";DATE$;
    ",";DESS$,",";AMT
280 CLOSE 2
290 CLOSE 15
300 LET I=I + 1
310 GOTO 120
320 REM SUM THE CHARGES
330 LET SUM=0
340 PRINT "NAME OF CARD TO TOTAL: ";
350 INPUT TEMP$
360 IF TEMP$="QUIT" THEN END
370 OPEN 15,8,15
380 OPEN 2,8,2,"CHARGE,L," + CHR$(125)
390 FOR K=1 TO I - 1
400 PRINT#15,"P"CHR$(2)CHR$(K)CHR$(0)CHR$(1)
410 GOSUB 1000
420 INPUT#2,CARD$,NAME$,DATE$,DESS$,AMT
430 IF CARD$ <> TEMP$ THEN 450
440 LET SUM=SUM + AMT
450 NEXT K
460 PRINT "TOTAL FOR "; TEMP$;" IS ";SUM
470 CLOSE 2
480 CLOSE 15
490 GOTO 320
1000 REM CHECK RECORD NOT PRESENT ERROR
1010 INPUT#15,ERRNUM,ERRNAMES
1020 IF ERRNUM=50 OR ERRNUM=0 THEN RETURN
1030 PRINT ERRNUM,ERRNAMES
1040 CLOSE 2
1050 CLOSE 15
1060 END

```

Note: This program does not accumulate the information on the file. You might want to provide for adding information to the file as was done in Example 2.






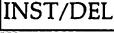

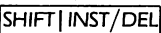

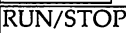


# INDEX

- Accessing modes
  - Upper case/graphics 16,238
  - Upper/lower case 16,238
- ABS 129,135,136
- Add Records Program 277-278
- Addition 11,16,42
- Animation program 242-243
- Antenna switchbox 10
- Arithmetic in BASIC 42,51-52,53-54
- Arithmetic Operators
  - order 51,52
  - priority 52
  - symbols 51
- Arithmetic priority 52
- Array Operations program 171-172
- Arrays 151,152-153,162-164,  
171-172,195-198
- ASC 129,135,136
- ASCII character set 188,192
- Automobile License Fees
  - program 106-111
- Averaging Numbers program
  - 111-112
- Bar Graphs program 241
- BASIC arithmetic 42,51-52,53-54
- BASIC functions
  - ABS 129,135,136
  - ASC 188,193
  - CHR\$ 187,193,237-238
  - INT 128,135-136,140
  - LEN 184,192
  - MID\$ 185,192
  - RND 248,253-254
  - SGN 129,135,136
  - SQR 127,135
  - STR\$ 189-190
  - TAB 71,72,76,127
  - VAL 189-190,191
- BASIC origins 6
- BASIC parentheses 45,53-54
- BASIC programs
  - entering and controlling 21,34
  - execution 33-34
  - retrieval 48,55-56
  - storage 47,55-56
- BASIC program requirements
  - line numbers 22,24,32-33
  - order 32-33
  - spacing 27-28,32
- BASIC statements
  - CLOSE 50,56,266,272-273
  - DATA 64-66,73-74
  - DEF 201,202-203,209-210,213
  - DIM 152,157-159,163-164
  - END 24,32
  - FOR NEXT 122,130-134
  - GET 98
  - GOSUB 205,210-212
  - GOTO 26,101-102
  - IF THEN 94,102-104
  - INPUT 26,28,37,62-64,73,82
  - INPUT# 267,274-275
  - LET 22,32,36-37,72
  - ON...GOSUB 208,212,281,283
  - OPEN 266,272-273
  - PRINT 11,16-17,30,32,67-68,  
71,74-77
  - PRINT# 266,273-275
  - READ 64-66,73-74
  - REM 68-70,77
  - RETURN 205,210-212
- BASIC variables, names 34-37

- Birthday Pairs in a Crowd program 259-261
- BREAK IN 34
- Carpet Estimating program 214-219
- CHANNEL SELECTOR Switch 10,15
- Channel Selector, TV 10
- Character strings 17,36
- CHR\$ 187,193,237-238
- CLEAR/HOME 12
- CLOSE 50,56,266,272-273
- Color 229,233,238-239
- Color Bar Graphs program 241-242
- Commands
  - END 24,32
  - LIST 21,33
  - LOAD 48,49,56
  - LOAD "\$",8 48,56
  - OPEN 266,284
  - OPEN 15,8,15 47,50,55,56,266,272
  - RUN 21,33,34
  - SAVE 48,56
  - SCRATCH 50,56
- Commodore **C=** key 11,15,16
- Commodore 64 BASIC 6
- Commodore 64 User's Guide 2,10
- Conditional transfer 93,102,140-141
- Converting Temperature program 80-81
- Correcting mistakes 10,17-18,31-33
- Course Grades program 168-171
- Crossed loops 133-134
- CTRL 229,237,238,239
- Cursor 11,29,236-237
- Cursor Movement program 239-241
- Cursor Placement Subroutine 230,236-237
- DATA 64,73-74
- DEF 201,202-203,209-210,213
- Deleting Lines 24,33
- DELETE key 12
- Depreciation Schedule program 141-144
- DIM 152,157-159,163-164
- Direct mode 16
- Directory 48,56
- Discovery strategy 5
- Disk Drive 10,42,47
- Distribution of Random Numbers program 256-257
- Division 12,16,42
- Editing lines 10,17-18,31-33
- END 24,32
- E Notation 54-55
- Error correction 10,17-18,31-33
- Error messages
  - BAD SUBSCRIPT 153
  - BREAK IN 34
  - EXTRA IGNORED 73
  - NEXT WITHOUT FOR 127
  - OUT OF DATA 66
  - SYNTAX ERROR 12
  - ?? 73
- Exact Division program 140-141
- Examination Grades program 165-168
- Exponentiation 45-46,51
- Files 266-267,272-273
- Finding the Average of a Group Numbers program 137-138
- Flipping Coins program 254-255
- FOR NEXT 122,130-134
- Formatting a Diskette 47,55
- GET 98
- GOSUB 205,210-212
- GOTO 26,102
- Graphics characters 232,237-238
- Greater than 96,104
- IF THEN 94,102-104
- Information path 266,267,272
- Initializing a diskette 47,55

INPUT 26,28,37,62-64,73,82  
 INPUT# 267,274-275  
 INST/DEL key 12,29  
 Inserting lines 25,32  
 INT 128,135-136,140  
 Interrupting program executions 26,  
 27,28,34  
 Interrupting INPUT loops 26,27,28,34  
  
 Jumping out of loops 26,27,28,34  
 Jumping out of INPUT loops  
 26,27,28,34

#### Keys

 11,15,16  
 12,229  
 29,229-230  
 29,229-230  
 229,237,238,239  
 12,29,31  
 12,22,25,34,  
 229-230  
 29,31  
 26,27,28,34  
 26,27,28,34  
 16,26,27,  
 34,230,233,239  
 12

LEN 184,192  
 Less than 96,104  
 LET 21,34-36,37  
 Line deletion 24,33  
 Line numbers 32  
 LIST 21,23,24,33,34  
 LOAD 48,49,56  
 LOAD "\$",8 48,56  
 Logical expressions 235-236  
 Loops, crossed 133-134  
 Loops, FOR NEXT 122,130-134  
 Loops, INPUT 26,28

Mail List Data Entry program  
 276-277  
 Mailing Label program 278-279  
 Matrix 152  
 Mean 221  
 Menu-Driven Mailing program  
 282-283  
 MID\$ 185,192  
 Modifying the Mail List  
 program 280-282  
 Multiplication 12,16,42  
  
 NEW 21,24,33,36  
 Not equal to 104  
 Numeric variable definition 34-37  
  
 ON/OFF switch 10  
 ON...GOSUB 208,212,281,283  
 OPEN 266,272-273  
 OPEN 15,8,15 47,50,55,56,266,  
 272  
 Operations  
   Arithmetic 16,51-52  
   Order 51-52  
   Order (of arithmetic operations)  
   51-52  
  
 Parentheses 16,53-54  
 Precision Record Access program  
 284-285  
 PRINT 11,16-17,30,32,67-68,71,  
 74-77  
 PRINT# 266,273-275  
 Printout of Number Patterns program  
 104-106  
 Priority of operations 51-52  
 Programs in book  
   Add Records program 277-278  
   Animation 242-243  
   Array Operations 171-172  
   Automobile License Fees 106-111  
   Averaging Numbers 111-112  
   Bar Graphs 241  
   Birthday Pairs in a Crowd 259-261  
   Carpet Estimating 214-219

- Color Bar Graphs 241-242
- Converting Temperature 80-81
- Course Grades 168-171
- Cursor Movement 239-241
- Depreciation Schedule 141-144
- Distribution of Random Numbers 256-257
- Exact Division 140-141
- Examination Grades 165-168
- Finding the Average of a Group of Numbers 137-138
- Flipping Coins 254-255
- Mail List Data Entry program 276-277
- Mailing Label program 278-279
- Menu-Driven Mailing program 282-283
- Modifying the Mail List program 280-282
- Precision Record Access 284-285
- Printout of Number Patterns 104-106
- Random Integers 256
- Random Colors 259
- Random Walk 257-258
- Replacement Code 194-195
- Rounding Off Dollar Values to Cents 212-214
- Sales Chart as a String Array 195-197
- Selected Labels program 279-280
- String Reversal 193-194
- Sum and Product of Numbers 81-83
- Temperature Conversion Table 139-140
- Unit Prices 78-80
- Word Count 194
- Random Integers program 256
- Random Colors program 259
- Random numbers 248
- Random Walk program 257-258
- READ 64,70-71,73-74
- Record 266,272
- REM 68-70,77
- Replacement Code program 194-195
- Reserve words 17
- Restarting BASIC 34
- RESTORE key 16,26
- RETURN 10,11,16,34
- Retrieving BASIC programs 48,55-56
- RND 248,253-254
- Rounding Off Dollar Values to Cents program 212-214
- RUN 21,23,33-34
- RUN/STOP key 16,26,27,28,34
- RUN/STOP|RESTORE key 16,26,27,34,233,239
- Sales Chart as a String Array program 195-197
- SAVE 47,55-56
- SCRATCH 50-56
- Selected Labels program 279-280
- SGN 129,135,136
- SHIFT key 10,11
- SHIFT|CLR/HOME 12,22,25,34
- SHIFT|INST/DEL key 29,31
- SHIFT LOCK key 15
- Spacing of printout 71,75-77
- SQR 127,135
- Standard deviation 221
- Storing BASIC programs 47,55-56
- String output 74,191
- String Reversal program 193-194
- String variable definition 36
- STR\$ 189-190
- Subscripts 152-153,161-163,164
- Subroutines 202,207,210-212
- Substrings 185,192
- Subtraction 16,42
- Sum and Product of Numbers program 81-83
- Switches
  - Antenna switchbox 10
  - CHANNEL SELECTOR 10
  - ON/OFF (rocker switch) 10
  - TV Channel Selector 10,15
- SYNTAX ERROR 12

TAB 71,72,76

Temperature Conversion Table  
program 139-140

Tracing programs 101

Transfer

conditional 93,96,102-104

unconditional 27,93,  
101-102

Troubleshooting programs 93,113

Turning on/off your computer 10

TV Channel Selector switch 10

Unconditional transfer 27,93,101-  
102

Unit Prices program 78-80

Upper case/graphics mode 16,238

Upper/lower case mode 15,238

VAL 189-190,191

Variable names 34-37

Variables, subscripted 161-163

Word Count program 194

+ 10,11,42,191

- 10,11,42

/ 12,42

\* 12,42

= 35,104,226,235

↑ 10,44-46,51

■ 11

< 94,96,104

> 104

<= 104

>= 95,96,104

<> 104

# NOTES



# NOTES

# QUICK REFERENCE GUIDE

## COMMANDS

**CONT** Restarts execution after use of the **RUN/STOP** key

**Deleting a file:** See the **OPEN** command

**LIST** Displays the program in memory

**LIST** Lists the entire program

**LIST 40-310** Lists lines 40 through 310

**LIST -400** Lists all lines through 400

**LIST 400-** Lists all lines from 400 to the end

**LOAD"filename",8** Brings the specified program from diskette into memory

**LOAD"MENU",8**

**NEW** Erases the program in memory

**OPEN 15,8,15,"NEW0:diskname, id"** Formats a diskette with the given diskname and id; destroys any information previously on the diskette

**OPEN 15,8,15,"NEW0:TAXFILES,ZZ"**

**OPEN 15,8,15,"SCRATCH0:filename"** Deletes a file from the diskette

**OPEN 15,8,15,"SCRATCH0:TAXJUNE"**

**RUN** Performs the program in memory

**SAVE"filename",8** Writes the program in memory onto the diskette and gives it the specified name

**SAVE"LABELS",8**

**SAVE"@filename",8** Throws away the contents of the old specified file and replaces them with the program in memory

**SAVE"@LABELS",8**

## FUNCTIONS

**ABS(number)** Computes absolute value

**LET X = ABS(-7)**

**ASC(string)** Returns the ASCII code of the first character of the specified string

**LET A = ASC(B\$)**

**CHR\$(number)** Takes an ASCII code number and returns the corresponding character

**PRINT CHR\$(65)**

**INT(number)** Returns the greatest integer less than or equal to the specified number

**LET Y = INT(2.7)**

**LET D = INT(RND(1)\*100)**

**LEN(string)** Returns the length of a string

**FOR X = 1 TO LEN(A\$)**

**MID\$(string,n1,n2)** Returns a substring n2 characters long starting with the n1th character of the original string

**IF MID\$(A\$,7,4) = "NAME" THEN 300**

**RND(n)** Returns a random number between (and including) 0 up to (but not including) 1; a negative n reseeds the random number generator so subsequent calls to **RND** return the same sequence of random numbers

**LET D = 1 + RND(1)\*9**

**IF RND(-2) > .5 THEN 220**

**SGN(number)** Returns the sign of the specified numeric expression

**LET P = SGN(R - E)**

**SQR(number)** Returns the string representation of a number

**LET N\$ = STR\$(27)**

**TAB(n)** Not a proper function, but used with the **PRINT** statement; moves the cursor to the specified tab position

**PRINT "HELLO";TAB(12);N\$**

**VAL(string)** Converts a string of digits into a number

**LET P = VAL(A\$)**

## STATEMENTS

**CLOSE n** Closes the specified open file  
**CLOSE 1**

**DATA** datalist Contains values to be assigned to variables in a **READ** statement  
**DATA 7.2,GIRL,3.4,5**

**DEF FNv1(v2)** Defines a function, named FNv1, of the variable v2  
**DEF FN(X) = X^2 - 5**

**DIM v(n)** Reserves space for arrays  
**DIM A(10,20), B\$(50)**

**END** Stops program execution

**FOR . . . TO/NEXT** Creates a loop  
**FOR J = 1 TO 10**  
**NEXT J**  
**FOR A = 2\*X TO Y STEP 2**  
**NEXT A**

**GET v\$** Assigns the last key pressed to v\$  
**GET A\$**

**GOSUB line #** Calls a subroutine beginning at specified line number  
**GOSUB 500**

**GOTO line#** Jumps to specified line number (unconditional jump)  
**GOTO 412**

**IF condition THEN line#** Jumps to the line number specified after **THEN** when the condition is true (conditional jump)  
**IF X>10 THEN 320**

**IF condition THEN statement** Performs the statement after **THEN** when the condition is true  
**IF Y-2=7 THEN END**

**INPUT v** Causes program to pause for input from the keyboard and assigns the input to the specified variables  
**INPUT A,B INPUT D\$**

**INPUT#n,v** Causes information to be read from a file on diskette and assigned to the specified variables  
**INPUT#1,A,N\$**

**LET v = expression** Assigns a value to a variable  
**LET X = 7 LET C = C + 1**

**ON v GOSUB n1,n2, . . .** Causes a jump to one of several subroutines depending on the value of the variable after the word **ON**  
**ON N GOSUB 300,350,400**

**ON v GOTO n1,n2, . . .** Causes a jump to one of several line numbers depending on the value of the variable after the word **ON**  
**ON Y3 GOTO 100,200,245**

**OPEN 15,8,15** Opens a location information path; required before other file operations

**OPEN n1,8,n1,"filename,L,"+CHR\$(n2)** Opens information path n1 for disk input or output operations, assigns it to the specified filename, and sets the record length of that file on n2  
**OPEN 2,8,2, "DATA,L,"+CHR\$(50)**

**PRINT list** Causes output to be placed on the specified output device; default is the screen  
**PRINT A,B PRINT C\$,X(I)**  
**PRINT#1,A\$**

**READ variablelist** Reads the next item in the **DATA** list and assigns it to a specified variable  
**READ B\$,NUMBER,M(K)**

**REM** Allows insertion of a comment in a program line  
**REM SUBROUTINE SCALE**

**RESTORE** Resets data in **DATA** statements so it can be read again

**RETURN** Returns program control from a subroutine to the line following the **GOSUB** that called it

## Selected ASCII Codes

Code	Effect	Usage
18	Turn reverse character mode on	PRINT CHR\$(18)
19	Move cursor to HOME position	PRINT CHR\$(19)
29	Move cursor right one position	PRINT CHR\$(29)
146	Turn reverse character mode off	PRINT CHR\$(146)
147	Clear screen and move cursor to HOME position	PRINT CHR\$(147)
154	Print characters in light blue (the start-up color)	PRINT CHR\$(154)
157	Move cursor down one line	PRINT CHR\$(157)

## SPECIAL KEYS

### Cursor Control and Editing

**←CRSR→** Moves the cursor right one space

**↑CRSR↓** Moves the cursor down one line

**SHIFT ←CRSR→** Moves the cursor left one space

**SHIFT ↑CRSR↓** Moves the cursor up one line

**INST/DEL** Deletes the character to the left of the cursor

**SHIFT INST/DEL** Inserts a space to the left of the cursor

**RETURN** Signifies end of current line; when editing, accepts edit changes that were made in the line

### Other Special Keys

**CLR/HOME** Moves the cursor to the HOME position in the upper corner of the screen

**SHIFT CLR/HOME** Clears the screen and places the cursor in the upper left corner (HOME) position

**CTRL [0] . . . CTRL [9]** Change character color or mode as indicated on the fronts of the number keys

**SHIFT [C=]** Switches between the upper case/lower case character modes; ([C=] is the Commodore Key)

**SHIFT** Accesses the graphic characters on the left side of the key fronts when in the upper case/graphics mode

**RUN/STOP** Interrupts a BASIC program

**RUN/STOP RESTORE** Stops anything in progress, erases the screen, and returns control to the command level

### Graphics Characters

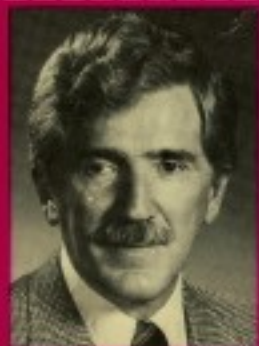
Are accessed with the **SHIFT** and **[C=]** keys in upper case/graphics mode; the characters are on the fronts of the keys; (see above)



HANDS-ON BASIC  
FOR THE  
COMMODORE 64

PECKHAM

McGraw-Hill



### About the Author

HERBERT PECKHAM is a principal in Computer Literacy, a firm specializing in materials for computer education. He formerly was professor of mathematics and physics at Gavilan College where he also taught courses in computing languages.

The author has served on the Executive Board of the American Association of Physics Teachers and was a member of that organization's committee on computers in physics teaching. Professor Peckham is author and co-author of numerous books and monographs on BASIC, Pascal, and computer applications.

When using, fold flap  
inside back cover.

# Hands-On BASIC

FOR THE COMMODORE 64

### About the Book

**Hands-On BASIC for the Commodore 64** is one of a series of adaptations of an earlier work by the author. This version includes sections on graphics and files. The book makes use of the hands-on method, providing computer experience through a series of guided activities. Each of these activities is followed by a discussion of the BASIC topic just explored. The hands-on method is a proven, efficient way to learn BASIC programming with a minimum of supervision.

EVITE EDITIONS



ISBN 0-07-049154-2